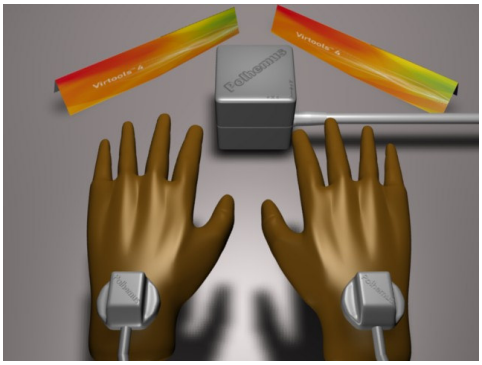
INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa



Concepção e Desenvolvimento de um jogo 3D para Computador utilizando o *software VIRTOOLS* e Tecnologia de Seguimento

Ricardo de Castro Aguiar, nº49482

Dissertação para obtenção de Grau de Mestre em Mestrado em Engenharia Electrotécnica de Computadores
Relatório de Tese de Mestrado
412/2007/M

Presidente: Mário Serafim dos Santos Nunes
Orientador: João António Madeiras Pereira
Co-Orientador: Carlos Jorge Ferreira Silvestre
Vogal: Manuel João Toscano Próspero dos Santos

Setembro 2007

Agradecimentos:

Esta tese não teria sido possível sem a ajuda das seguintes pessoas, os meus pais Daniel e Suzete Aguiar e irmãos Paulo e Rafael Aguiar que ao longo de toda a minha vida têm-me dado o seu apoio incondicional, Prof João M. Pereira que aceitou orientar a minha tese dando-me assim a possibilidade de trabalhar numa área de eleição para mim, a área dos videojogos, Prof Carlos Silvestre que aceitou co-orientar a minha tese e assim criou a ponte necessária entre o Departamento de Engenharia Electrotécnica e Computadores e o Departamento de Engenharia Informática e de Computadores, Eng. Bruno Rodrigues de Araújo por ajudar a dar um rumo à minha tese, pelas dicas acerca dos aspectos relacionados com a interface das aplicações e testes de utilizador, colegas Luís Tavares, Nelson Gonçalves e Ricardo Ferreira pelas dicas na área de Sistemas de Decisão e Controlo, pelo beta-testing, almoços e principalmente pelas "Reuniões das 17h" no ISR, colega Luís Loureiro pela ajuda e conselhos de modelação em *Blender* e pelo *beta-testing* das aplicações, Diogo Neves pelo *beta-testing* das aplicações, Luís Coutinho pela ajuda na área musical, André Serpa por filmar as aplicações e pelo *beta-testing* e finalmente um grande obrigado a todos os elementos do Fórum *Swap Meet* [1].

Resumo:

A presente tese tem a finalidade de criar uma aplicação interactiva 3D, utilizando a ferramenta profissional de desenvolvimento de videojogos *Virtools*, que apele à destreza e astúcia do utilizador através do uso de diversos dispositivos imersivos estado de arte, tais como luvas de captura de movimentos, óculos de realidade virtual e *tracker* magnético.

Adicionalmente é utilizada o *software Microsoft Visual Studio.NET* de modo a desenvolver novas funcionalidades na ferramenta acima referida.

Palavras-Chave:

Realidade Virtual, Dispositivos Imersivos, Motor de Jogo, Jogo de Computador, Interactividade, Imersão.

Abstract:

This thesis has the goal to create an interactive 3D application with Virtools, a professional videogames developer tool, that appeal to the gamer dexterity and astuteness through the use of several immersive devices like data gloves, VR glasses and magnetic trackers.

Microsoft Visual Studio.Net platform is also used to develop new plug-ins into Virtools.

Keywords:

Virtual Reality, Immersive Devices, Game Engine, Computer Game, Interactivity, Immersion.

Índice:

Agradecimentos: iii

Palavras-Chave: v

Resumo: v

Abstract: vii

Keywords: vii

Índice: ix

Lista de Figuras: xiii

Lista de Tabelas: xvii

Lista de Siglas: xix

1 Introdução 1

1.1 Enquadramento 1

1.2 Organização do documento 1

1.3 Dispositivos de Integração para Realidade Virtual 2

1.3.1 Dispositivos de entrada 2

1.3.2 Dispositivos de saída 4

2 Tecnologias utilizadas 5

2.1 Dispositivos imersivos de entrada 5

2.1.1 *Polhemus Fastrak* 5

2.1.2 Óculos virtuais *eMagin Z800 3D Visor* 9

2.1.3 Luvas *5DT* 9

2.2 Dispositivos imersivos de saída 10

2.2.1 Óculos Virtuais *eMagin Z800 3D Visor* 10

2.3 *Software* de Desenvolvimento 12

2.3.1 *Dassault Systèmes Virtools* 12

3 Integração de Dispositivos Imersivos em *Virtools* 17

3.1 *Polhemus Fastrak* 17

3.2 Luvas *5DT* 20

3.3 Óculos virtuais *eMagin Z800* 22

3.3.1 Seguimento 23

3.3.2 Visualização 24

4 Conceitos básicos para a criação de jogos em *Virtools* 31

4.1 Criação e controlo de câmara 31

4.2 Aparência 32

4.2.1 Modelo de reflexão de *Phong* 32

4.2.2 Modelo de iluminação de *Gouraud* 33

4.3 Transformações geométricas 34

4.4 Detecção de Colisões 37

4.5 Motor de Física 39

4.6 Efeitos especiais 40

4.6.1 Sombras *Stencil*: 41

4.6.2 Reflexões planares 44

4.6.3 Sistemas de Partículas 45

5 Desenvolvimento de Aplicações de Realidade Virtual 47

5.1 *PongVR* 47

5.1.1 Aquisição e tratamento de dados de gestores de dispositivos 49

5.1.2 Motor físico de jogo 50

5.1.3 Gestão de lógica de jogo 50

5.1.4 Cálculo de Inteligência Artificial de adversário 53

5.2 *PianoVR* 57

5.2.1 Aquisição e tratamento de dados de gestores de dispositivos 58
5.2.2 Gestão de detecção de colisões 60
5.2.3 Gestão de reprodução de sons 61
5.2.4 Gestão de lógica de jogo 61
5.2.5 Gestão de temporizador 61
6 Medição de Desempenho 63
6.1 *Polhemus Fastrak* 65
6.2 Luvas *5DT Ultra* 66
6.3 Óculos *eMagin Z800 3D Visor* 66
7 Testes de Utilizador 67
7.1 Objectivos 67
7.2 Descrição do sistema a testar 67
7.3 Ambiente de teste 67
7.4 Características dos participantes 67
7.5 Metodologia 67
7.6 Tarefas 68
7.7 Testes e Medidas 68
7.8 Conclusões para Testes de Utilizador 70
7.8.1 *PongVR* 70
7.8.2 *PianoVR* 70
8 Conclusões 71
9 Referências 73
10 Bibliografia 75
11 Anexos 1
11.1 Anexo A – Manual de Instruções de jogo *PongVR* 1

11.2 Anexo B – Manual de Instruções de jogo PianoVR 7

11.3 Anexo C - Apresentação *Virtools* 12

11.4 Anexo D – Scripts PongVR 20

11.5 Anexo E – Scripts PianoVR 23

Lista de Figuras:

Figura 2-1 Componentes *Polhemus Fastrak* 5

Figura 2-2 Uma bobine.....6

Figura 2-3 Três Bobines 7

Figura 2-4 Ligação Luvas *5DT* 9

Figura 2-5 Mapeamento luva 10

Figura 2-6 *Anaglyph Stereo* 11

Figura 2-7 *Passive Stereo* 11

Figura 2-8 *Active Stereo* 12

Figura 2-9 *Layout* principal *Virtools* 12

Figura 2-10 Exemplo blocos de comportamentos/*Behavioural Building Blocks* 13

Figura 2-11 Componentes de um bloco de comportamento 14

Figura 2-12 Localização de *Start*, *bLink*, *pLink*, *shortcut pLink*.....14

Figura 2-13 *Drag & Drop* de Recursos 15

Figura 2-14 *Drag & Drop* de blocos de comportamentos.....15

Figura 3-1 Bloco de comportamento *TrackerBB* 17

Figura 3-2 Funcionamento *TrackerBB* (à esquerda) + Gestor (à direita).....19

Figura 3-3 Má utilização espacial do tracker magnético 19

Figura 3-4 Funcionamento de *GloveBlock* (à esquerda) + Gestor (à direita) 20

Figura 3-5 Bloco de comportamento *GloveBlock* 21

Figura 3-6 Mapeamento de gestos 22

Figura 3-7 Bloco de comportamento *eMaginBB* 23

Figura 3-8 Funcionamento de *eMaginBB* (à esquerda) + Gestor (à direita).....24

Figura 3-9 Definição de câmeras 25

Figura 3-10 Esquema correcto de estereoscopia.....25

Figura 3-11 Estereoscopia com câmara convencional 26

Figura 3-12 Estereoscopia com *Projection Referential Cameras*.....26

Figura 3-13 Conteúdo de *VRDisplay.cfg*.....27

Figura 3-14 Conteúdo de *VRStereo.cfg*.....27

Figura 3-15 Projecção de pontos no plano de visualização.....28

Figura 3-16 Tipos de *paralax*.....29

Figura 4-1 Projecção Ortográfica.....31

Figura 4-2 Projecção Perspectiva.....32

Figura 4-31 Iluminação por pixel vs iluminação por vértice.....34

Figura 4-4 Sistema de coordenadas regra da mão esquerda.....35

Figura 4-5 Translação.....35

Figura 4-6 Rotação.....35

Figura 4-7 Ângulos de Euler.....36

Figura 4-8 Escala.....37

Figura 4-9 Volume Esfera.....37

Figura 4-10 Volume *AABB*.....38

Figura 4-11 Volume *OBB*.....38

Figura 4-12 Restrições em motores de física.....39

Figura 4-13 Sistema de *ragdolls*.....40

Figura 4-14 Volume de Sombra.....42

Figura 4-15 Modificação Stencil Buffer 1º algoritmo, 1 objecto.....42

Figura 4-16 Modificação Stencil Buffer 1º algoritmo, múltiplos objecto.....43

Figura 4-17 Erro no preenchimento de Stencil Buffer.....43

Figura 4-18 Preenchimento de Stencil buffer.....44

Figura 4-19 Reflexões planares.....44

Figura 4-20 Sistema de partículas.....46

Figura 5-1 *PongVR*.....47

Figura 5-2 Arquitectura simplificada de *PongVR* 48

Figura 5-3 Pseudo-código de scripts de aquisição de dados de PongVR 49

Figura 5-4 Máquina de estados de atribuição de pontuação em *PongVR* (1) 51

Figura 5-5 Máquina de estados de atribuição de pontuação em *PongVR* (2) 51

Figura 5-6 Máquina de estados de atribuição de pontuação em *PongVR* (3) 52

Figura 5-7 Áreas de jogo 52

Figura 5-8 Contador de toques consecutivos 53

Figura 5-9 Localização marca – Nível de dificuldade médio 54

Figura 5-10 Localização Marcas - Nível de dificuldade difícil 54

Figura 5-11 Aspecto *in-game* de *PongVR* com ajudas visuais activas 56

Figura 5-12 *PianoVR* 57

Figura 5-13 Arquitectura simplificada de *PianoVR* (1) 57

Figura 5-14 Arquitectura simplificada de *PianoVR* (2) 58

Figura 5-15 Arquitectura simplificada de *PianoVR* (3) 58

Figura 5-16 Mapeamento de ossos na mão virtual 59

Figura 5-17 Pseudo-código de aquisição de informação de flexão das luvas 60

Figura 5-18 Colisão dedo-tecla 61

Figura 6-1 Funcionamento de *Get_Precise_Time* (à esquerda) + Gestor (à direita) 63

Figura 6-2 Aspecto visual de Get System Time e Get_Precise_Time 64

Anexo - Figura 1 *PongVR Main Menu* 2

Anexo - Figura 2 *PongVR Start Menu* 2

Anexo - Figura 3 *PongVR In Game* 3

Anexo – *Figura 4 PongVR Options Menu* 3

Anexo - Figura 5 *PongVR Help Menus* 5

Anexo – Figura 6 *PongVR Credits Menu* 6

Anexo - Figura 7 PianoVR Main Menu 8

Anexo - Figura 8 *PianoVR Start Menu* 8

Anexo - Figura 9 *PianoVR In Game* 9

Anexo - Figura 10 *PianoVR Options Menu*.....9

Anexo - Figura 11 *PianoVR Help Menus*.....11

Anexo - Figura 12 *PianoVR Credits Menu*.....12

Anexo - Figura 13 Apresentação *Virtools* 18

Anexo - Figura 14 Script de comunicação com Gestores 20

Anexo - Figura 15 Script de tratamento de dados.....20

Anexo - Figura 16 Conteúdo grafo comportamental 'Implement Right Hand Rotation' 21

Anexo - Figura 17 Movimentação de objecto após inclusão no motor de física.....21

Anexo - Figura 18 Conteúdo grafo comportamental 'Implement Right Hand Motion' 22

Anexo - Figura 19 Script de configuração de estereoscopia 22

Anexo - Figura 20 Grafo comportamental 'Implement Left Finger Motion'.....23

Anexo - Figura 21 Script de tratamento de dados de mão esquerda.....23

Anexo - Figura 22 Alinhamento mão direita.....23

Lista de Tabelas:

Tabela 1-1 Comparação de tecnologias de *tracking* [6], [7], [8] 3

Tabela 1-2 Comparação de tecnologias de *Data Gloves*, consultar [9], [10] 4

Tabela 2-1 Comparação modelos *5 Ultra* e *14 Ultra* 10

Tabela 3-1 Identificação de sensores activos 21

Tabela 5-1 Mapeamento ossos-sensores para luva mão esquerda 59

Tabela 5-2 Mapeamento ossos-sensores para luva mão direita 60

Tabela 6-1 Desempenho de dispositivos imersivos 65

Tabela 7-1 Questionário *PongVR* 69

Tabela 7-2 Questionário *PianoVR* 69

Lista de Siglas:

3D	Três Dimensões
5DT	*Fifth Dimension Technologies*
AABB	*Axis Aligned Bounding Box*
A/D	*Analog/Digital*
API	*Application Programming Interface*
BB	*behavioural Building Block*
bIn	*Behaviour Input*
bLink	*Behaviour Link*
bOut	*Behaviour Output*
BSP	Binary Space Partitioning
CPU	Central Processing Unit
CRT	*Cathode Ray Tube*
DoF	*Degrees of Freedom*
FPS	*Frames Per Second*
HUD	*Heads Up Display*
Hz	*Hertz*
INESC	Instituto de Engenharia de Sistemas e Computadores
LCD	*Liquid Cristal Display*

MB	*Mega Byte*
MoCap	*Motion Capture*
OBB	*Oriented Bounding Box*
OLED	*Organic Light Emitting Diod*
PC	*Personnal Computer*
PCS	*Product Context Scenario*
pIn	*Parameter Input*
pOut	*Parameter Output*
RAM	*Random Access Memory*
RBGA	*Red Green Blue Alpha*
SDK	*Software Development Kit*
SDL	*Specification and Description Language*
USB	*Universal Serial Bus*
UTC	*Universal Time Co-ordinated*
VR	*Virtual Reality*
VRAM	*Vídeo RAM*
Wifi	*Wireless Fidelity*

1 Introdução

1.1 Enquadramento

Os jogos de computador e consolas são hoje em dia um mercado lucrativo, e são estes que ditam o avanço tecnológico do *Hardware* nos computadores pessoais e consolas de jogos. Cada vez é preciso maior capacidade de processamento num jogo, ora para simular efeitos físicos de extrema complexidade ou então para oferecer uma qualidade visual que torna a experiência de jogo cada vez mais realista. É assim apelativo para um programador enveredar pela criação de videojogos, criar um jogo que forneça ao jogador uma completa imersão numa nova realidade. No entanto, os jogos ditos convencionais estão quase sempre limitados aos dispositivos de entrada comuns, como o rato e teclado num computador ou um *gamepad* numa consola. Ainda mais, está-se agora a viver na era do 3D, Três Dimensões, em que o mundo virtual criado pelos videojogos apela às três dimensões do espaço através da utilização de técnicas de projecção perspectiva. Todavia o jogador é confrontado com o problema de que um jogo é visualizado numa tela de ecrã que apenas contem duas dimensões retirando a sensação de imersão a 100%.

Nos dias de hoje é igualmente hábito no desenvolvimento de videojogos a atenção desmedida na componente gráfica em detrimento das outras componentes que são igualmente importantes. Cenários e personagens extremamente detalhados são, sem dúvida, um bom passo mas não garantem o sucesso de um jogo. Infelizmente a componente de jogabilidade tem sido desprezada tornando os jogos pouco intuitivos e, apesar da boa aparência, pouco estimulantes a serem jogados. Felizmente já se começa a tomar consciência de que a jogabilidade é tão ou mais importante que a componente visual de um jogo, já se começam a explorar novas formas de interacção com o utilizador.

A presente tese visa o desenvolvimento de jogos utilizando uma ferramenta profissional, *Virtools*, que apele a formas de interacção de um jogador com um jogo recorrendo a dispositivos imersivos estado de arte. O objectivo é proporcionar ao jogador uma maior imersão e interacção no ambiente virtual de jogo sem recorrer a um rato, teclado e monitor.

1.2 Organização do documento

A presente dissertação é constituída por onze capítulos:

1. **Introdução:** É feita um enquadramento da tese seguido de uma pequena descrição de alguns dispositivos imersivos estado de arte existentes no mercado.
2. **Tecnologias utilizadas:** Neste capítulo é feita uma descrição detalhada das tecnologias utilizadas pelos dispositivos imersivos e *software* empregados na tese.
3. **Integração de dispositivos imersivos em *Virtools*:** Para a utilização dos diversos dispositivos na plataforma *Virtools* é necessário criar mecanismos de comunicação entre ambos, pelo que o processo de integração é aqui descrito.

4. **Conceitos básicos para criação de jogo em *Virtools*:** São explicados alguns conceitos necessários para o desenvolvimento de aplicações gráficas interactivas em *Virtools*.
5. **Desenvolvimento de aplicações de Realidade Virtual:** São desenvolvidas para a tese duas aplicações que utilizam os diversos dispositivos imersivos. Neste capítulo é explicada a arquitectura para cada uma destas.
6. **Medição de Desempenho:** A cada dispositivo está associado um tempo de latência proveniente do cálculo dos diversos dados associados e respectivos tempos de comunicação entre estes e o computador. Neste capítulo são mostradas medições temporais feitas e apresentadas conclusões sobre estas.
7. **Testes de Utilizador:** As aplicações são testadas por um grupo de indivíduos de modo a aferir o desempenho dos diversos dispositivos e aplicações. O resultado dos testes é analisado e são tiradas conclusões sobre estes.
8. **Conclusões:** São expostas neste capítulo as conclusões retiradas do desenvolvimento de todo o trabalho apresentado.
9. **Referências:** Contém referências utilizadas durante o desenvolvimento do trabalho
10. **Bibliografia:** Contém bibliografia usada durante o desenvolvimento do trabalho
11. **Anexos:** Neste capítulo é apresentada informação que seja importante referir, no entanto não completamente necessária para a compreensão do trabalho desenvolvido, tal como manuais de instruções e *scripts Virtools*.

Todos os esquemas de arquitecturas de aplicações e *plug-ins* são descritos em ***SDL***, ***Specification and Description Language*** [2], dado que é uma linguagem gráfica de descrição de *software* flexível e de fácil leitura e interpretação.

1.3 Dispositivos de Integração para Realidade Virtual

Nesta secção apresentam-se e descrevem-se dispositivos que proporcionam novos tipos de interacção de um utilizador com um computador ou consola relativamente às técnicas convencionais de com dispositivos como teclado, rato ou *gamepad*.

1.3.1 Dispositivos de entrada

1.3.1.1 Seguimento

- **Seguimento inercial: *Wii Remote***, também conhecido por ***Wiimote*** [3], [4], o comando da nova consola da *Nintendo*, a *Wii*, reforça o conceito de jogabilidade numa consola. Este comando é composto por sensores de movimento, acelerómetros nos 3 eixos, Vertical, Horizontal e Profundidade, que aliado a uma ligação sem fios *bluetooth* à consola dá ao utilizador a possibilidade de utilizar todo o seu espaço envolvente para, através dos movimentos aplicados ao comando, interagir com os jogos.

- **Seguimento Magnético:** ***Flock of Birds*** e ***Polhemus Fastrak*** [5] com ligação *USB, Universal Serial Bus,* ou porta serial *RS-232* a um computador, permitem saber a posição e orientação de um elemento receptor de campo magnético em relação a um elemento emissor de campo magnético. O funcionamento de ambos é semelhante. Nesta tese é utilizado o *tracker* ***Polhemus Fastrak*** pelo que é dada uma explicação aprofundada do seu funcionamento na secção **2.1.1** e **4.1**.
- ***Head Mounted Displays*****:** Óculos ***eMagin z800 3D Visor*** com giroscópio interno e acelerómetros, permite determinar a sua orientação e tipo de movimentos efectuados. Este dispositivo é utilizado nesta tese pelo que é dada uma explicação aprofundada na secção **2.1.2** e **4.3**.

É mostrada abaixo uma tabela em que são comparadas diversas tecnologias de seguimento através de diversos tempos de latência e frequências de amostragem. O tempo de latência indica o tempo necessário para fornecer os dados dos dispositivos ao computador a que estão ligados, e frequência de amostragem indica o número de amostras que são capturadas por segundo. Finalmente ***DoF*** indica o número de graus de liberdade que o respectivo dispositivo fornece ao utilizador.

Tabela 1-1 Comparação de tecnologias de *tracking* [6], [7], [8]

	Tempo de Latência	**Frequência de amostragem**	***DoF***	**Tecnologia**
InterGlide ™ IS-300	2 ms	125Hz- 500Hz	3	Inercial
FakeSpace, Inc. BOOM3C	200ms	>70Hz	6	Mecânico
Lipman: Vscope	2ms	100Hz	3	Ultra.som
DynaSight	16-31ms	64Hz	3	Óptico
Cyclop Tracker	<40ms	30-60Hz	6	Óptico
WorldViz Optical Tracker	20ms	60Hz	3	Óptico
Polhemus Fastrack	4ms	120Hz	6	Magnético
Flock of Birds	-	144Hz	6	Magnético

1.3.1.2 Captura de Movimentos

- ***Data Gloves***, dispositivos em forma de luva que se ligam a um computador via *USB* ou porta serial *RS-232*. Após calçadas, as luvas fornecem informação de flexão em pontos específicos da mão do utilizador. O detalhe da informação é determinado pelo número de sensores que estas contenham. Exemplos destes dispositivos são as luvas ***CyberGlove II*** [9], ou ***5DT*** [9]*, Fifth Dimension Technologies*. Estas últimas, de 14 e 5 sensores, são utilizadas nesta tese e como tal serão descritas em detalhe na secção **2.1.3**. e **4.2**.

  É de seguida apresentada uma tabela que visa a comparação entre diversos tipos de luvas de captura de dados.

Tabela 1-2 Comparação de tecnologias de *Data Gloves*, consultar [9], [10]

	Frequência de amostragem	Resolução
X-IST	60Hz	10bit
***MIDI Glove*]**	100-200Hz	10bit
CyberGlove II	90Hz	10bit
5DT Data Gloves	75Hz	10bit

Nos exemplos colocados acima nas tabelas, a documentação obtida para estes indica apenas a utilização de tensores mas não é encontrada especificação para o tipo de tensor em si.

1.3.1.3 3D Mouses

Dispositivos que substituem o rato convencional. Um ***3D Mouse*** fornece mecanismos adicionais que permitem a navegação intuitiva em ambientes 3D através do tipo de movimento aplicado a este. Exemplo de um *3D Mouse* é o caso do ***SpaceNavigator*** da empresa *3D Connection* [11].

1.3.2 Dispositivos de saída

1.3.2.1 Dispositivos Hápticos

Fornecem a sensação de tacto ao utilizador. Como exemplo desta tecnologia refere-se o dispositivo ***CyberGrasp***. Um *CyberGrasp* é uma luva de captura de movimentos que contém um **exo-esqueleto** associado à luva. O exo-esqueleto fornece um sistema de *force-feedback* a cada dedo de modo a que o utilizador possa sentir o tamanho e forma dos objectos *3D* no mundo virtual. Consultar [9].

1.3.2.2 Óculos de Realidade Virtual

Óculos que permitem uma melhor percepção da terceira dimensão perdida através da utilização de um monitor plano convencional (***Cathod Ray Tube***, ***CRT***, ou ***Liquid Crystal Dysplay***, ***LCD***). Os óculos poderão oferecer ao utilizador estereoscopia activa ou passiva. Para esta tese são utilizados os óculos ***eMagin z800 3D Visor*** [12] pelo que estes, conjuntamente com a noção de estereoscopia, serão explicados em maior detalhe na secção **2.2.1** e **4.3**.

2 Tecnologias utilizadas

No presente capítulo são detalhados os diversos dispositivos e plataformas utilizados no decorrer desta tese de mestrado. É dada uma descrição de funcionamento do *tracker* ***Polhemus Fastrak***, **Luvas *5DT 5/14 Ultra***, óculos ***eMagin Z800 3D Visor*** e *software* ***Virtools***. No capítulo seguinte será devidamente explicado como são retirados e tratados os diversos dados de cada um dos dispositivos e, como é feita a sua integração em duas aplicações desenvolvidas especificamente para a tese.

2.1 Dispositivos imersivos de entrada

2.1.1 *Polhemus Fastrak*

Dispositivo contém três componentes principais:

- **Módulo de Interface ou filtro**
- **Módulo emissor**
- **Módulo receptor**

Permite determinar posição e orientação, com seis graus de liberdade, de um elemento receptor em relação a um elemento emissor através de coordenadas cartesianas **(x,y,z)** e três **ângulos de *Euler***, estes serão detalhados no capítulo 3.

Nesta secção é descrito o funcionamento do *tracker Polhemus Fastrak* e são listados os pontos forte e fracos da utilização desta tecnologia em detrimento de outras com a mesma finalidade.

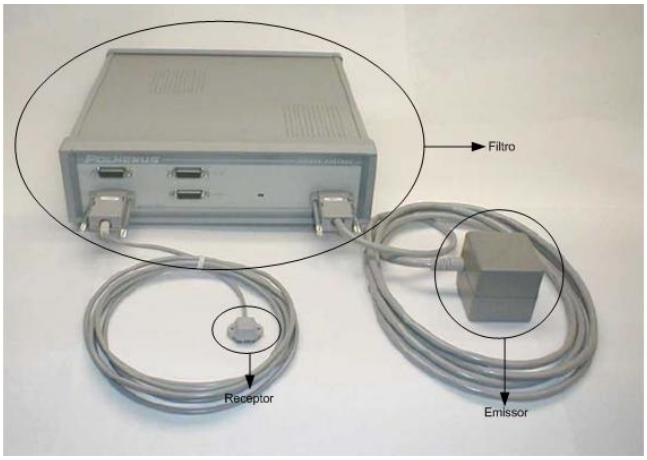


Figura 2-1 Componentes *Polhemus Fastrak*

- **Módulo de Interface, Filtro:** É no Módulo de Interface que é efectuada a ligação de um emissor e até quatro receptores. A ligação com o *Personnal Computer, PC,* é feita também através desse módulo, por uma interface *USB* ou *Serial*. O filtro é também responsável pelo tratamento dos dados recebidos pelos receptores de modo a serem gerados uma posição e orientação que os defina no espaço.

- **Emissor, Receptor:** Para se melhor compreender a obtenção de seis graus de liberdade do *tracker* será primeiramente explicado o caso de apenas um grau de liberdade, e a partir deste será expandido para os seis graus de liberdade.
    - **Um grau de liberdade:** Para fins exemplificativos é utilizado o eixo dos **yy's** para determinar a separação entre o emissor e o receptor. O emissor contem uma bobine, alinhada com o eixo dos **yy**, que será passada por uma corrente. A corrente tem o sentido apropriado para que seja gerado um campo magnético na direcção pretendida. A obtenção da direcção a tomar é obtida através da regra da mão direita. O receptor, por outro lado contem uma bobine igual, mas num estado passivo, ou seja, não irá ter numa corrente a ser injectada nesta. O campo magnético gerado pelo emissor irá criar uma corrente na bobine receptora proporcional à corrente que está a correr naquele e à distância entre ambos. Deste modo, dado que se sabe a corrente que está a ser utilizada no emissor, consegue-se determinar a distância entre o emissor e o receptor.

      Suponha-se agora que se pretende determinar a rotação num dos eixos do receptor em relação ao emissor. Com apenas uma bobine torna-se impossível obter essa informação, isto porque a intensidade do campo magnético no receptor é diminuída tanto pela distância com o emissor, como também pelo facto de as duas bobines não estarem paralelas. Assim é impossível determinar no caso do nosso campo magnético diminuir se é por aumento da distância ou por rotação do sensor.

Figura 2-2 Uma bobine

    - **Seis graus de liberdade:** O sistema completo para a determinação dos seis graus de liberdade é composto por três bobines ortogonais entre si, cada uma delas especificando um determinado eixo **[xx',yy',zz']**. A corrente que passa por cada uma das bobines é bem especificada. O receptor é também composto por um conjunto de três bobines. Cada um dos campos magnéticos gerados terá de ser isolado, e isso poderá ser feito ou através de **multiplexagem no tempo** ou **variando a intensidade da corrente aplicada**. Através da informação recolhida com a arquitectura especificada é possível determinar se a variação da corrente

nos receptores é devido a mudanças de orientação ou de distância entre emissor e receptor.

Figura 2-3 Três Bobines

Em cada uma das bobines, o tamanho das suas espiras é mantido muito pequeno em relação à distância entre o emissor e o receptor, isto para que o emissor e receptor possam ser considerados pontos ou dipolos infinitesimais. É produzido um campo magnético, excitando cada uma das bobinas, com uma componente longínqua e uma componente próxima, ***quasi-static***, esta que será predominante caso a distância entre receptor e emissor seja:

$$\rho << \frac{\lambda}{2\pi} \tag{2.1}$$

Em que λ é o comprimento de onda do sinal de excitação.

Assim, considerando uma corrente:

$$i(t) = I\cos(wt) \tag{2.2}$$

A atravessar uma bobine, produz-se um campo magnético a uma distância ρ e ângulo ς com as seguintes componentes:

$$H_\rho = (\frac{M}{2\pi\rho^3}) * \cos\varsigma \text{ - Componente radial de Campo Magnético} \tag{2.3}$$

$$H_t = (\frac{M}{4\pi\rho^3}) * \sin\varsigma \text{ - Componente tangencial de Campo Magnético} \tag{2.4}$$

Pelo que:

$$M = NIA \tag{2.5}$$

M é o momento magnético, **N** o número de espiras e **A** a área da bobine. Existem três bobines no receptor, para três momentos magnéticos pelo que se pode considerar o momento magnético total como a matriz:

$$M_t = [M_1 \mid M_2 \mid M_3]. \tag{2.6}$$

A posição e orientação do sensor é descrita através da tensão induzida nas três bobines, pela componente da ligação magnética e pelas suas sensibilidades descritas pela matriz:

$$S = [S_1 \mid S_2 \mid S_3]. \tag{2.7}$$

Assim nove valores de tensão são calculados através de:

$$V = \frac{K}{\rho^3} S^T M_t \tag{2.8}$$

Em que ***K*** é um factor de ganho. Acoplado à ligação magnética existe uma componente de ruído na ligação magnética em conjunto com ruído atmosférico representados por N_i e ruído electrónico representado por N_b. Estes factores terão de ser postos em conta no cálculo de ***V*** pelo é feita a soma algébrica das suas quantidades com a expressão acima.

A expressão para o cálculo das tensões fica com o seguinte aspecto final:

$$V = \frac{K}{\rho^3} S^T M_t + N_i + N_b \tag{2.9}$$

O módulo de interface tem aqui o papel fundamental, pelo que é neste que são feitos todos os cálculos e são descobertos os novos valores de posição e orientação através dos valores obtidos de ***V***. Consultar [5], [13], [14].

2.1.1.1 Pontos fortes

A tecnologia de *tracking* magnético tem, como qualquer tecnologia, pontos fortes e fracos. Aqui são listados os pontos que tornam esta tecnologia uma boa escolha para a captura de movimentos:

- Utilizando *trackers* magnéticos não há problemas de linha de vista. O campo magnético gerado atravessa obstáculos ligeiros. Assim pode-se utilizar os dados gerados pelo *tracker* mesmo que o receptor esteja ocluso em relação ao emissor
- O custo associado a um *tracker* magnético é inferior a um sistema de *tracking* óptico, *ex*. *MoCaP, Motion Capture*.
- A precisão obtida é bastante boa.
- Em circunstâncias ideais os *trackers* magnéticos podem ter um alcance de funcionamento elevado.
- O ritmo de amostragem apesar de não ser o maior, é dos maiores, com taxas a variarem nas **120 amostras/segundo com 1 receptor**, **60 amostras/segundo com 2 receptores**, **40 amostras/segundo com 3 receptores** e **30 amostras/segundo com 4 receptores**. Consultar documentação oficial *Polhemus* [5].

2.1.1.2 Pontos fracos

Infelizmente este tipo de tecnologia não está isenta de falhas. São listados alguns dos problemas que estão associados à utilização de um *tracker* magnético.

- O campo magnético gerado poderá receber interferências externas tais como dispositivos electrónicos nas redondezas, ou mobiliário com materiais condutores ou férreos.
- À medida que a distância vai aumentado entre o emissor e o receptor vai se perdendo precisão nos dados obtidos
- Dado que é necessário fazer processamento de dados no filtro, é introduzido aí um tempo de latência. Para uma aplicação com requisitos de tempo real muito exigentes pode-se tornar desastroso o tempo de latência no filtro.

- Poderão haver pequenas flutuações de corrente, o tipo de sala em que se está a utilizar o *tracker*, etc., fazendo com que este, apesar de estar parado, indicar movimento.

2.1.2 Óculos virtuais *eMagin Z800 3D Visor*

Os óculos *3D Visor* contêm uma componente de seguimento para além de uma componente de visualização. Estes possibilitam ao utilizador o seguimento dos movimentos feitos pela sua cabeça. Através de um sistema de giroscópios ***MEMS***, ***Micro-Electro Mechanical System***, de alta velocidade em conjunto com acelerómetros em cada um eixos **[xx',yy',zz']** são capturados todos os movimentos feitos pela cabeça do utilizador.

A tecnologia de *MEMS*, é descrita como a integração de diversos elementos mecânicos tais como sensores e actuadores num substrato comum de silício através de tecnologia de micro fabricação.

2.1.3 Luvas *5DT*

Através da utilização de luvas *5DT* é possível, através de sensores localizados em pontos específicos destas, determinar movimentos efectuados na mão em que a luva está calçada [15].

As luvas descritas nesta secção são os modelos ***Left Handed 5DT Data Glove 5 Ultra*** e ***Right Handed 5DT Data Glove 14 Ultra***. A principal diferença entre estes dois modelos é a localização e número de sensores que as constituem.

Cada luva contém um porto *Rj12* que, em conjunto com um cabo fornecido pela *5DT*, permite a ligação a um computador através de uma porta *USB* ou *Serial RS-232*. Existe igualmente disponível um módulo *Wifi, Wireless Fidelity,* que permite uma ligação sem fios, via *BlueTooth* entre o computador e as luvas, dando assim maior liberdade de manuseamento destas por parte do utilizador.

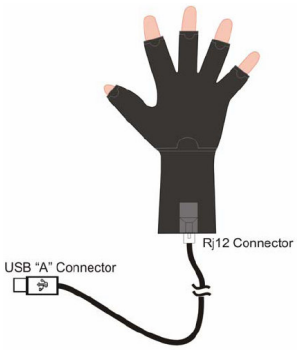


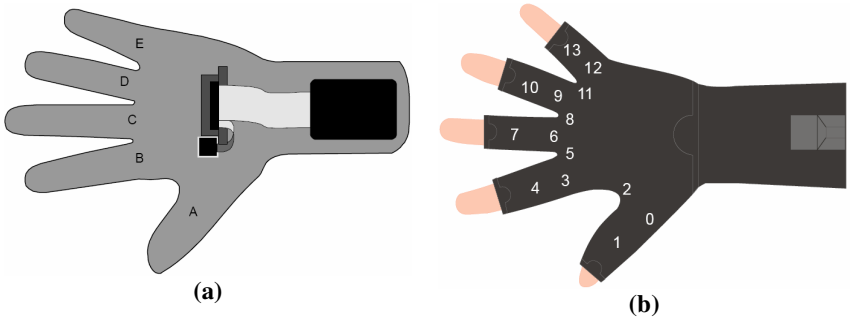
Figura 2-4 Ligação Luvas *5DT*

Abaixo é mostrada uma tabela comparativa entre os dois modelos.

Tabela 2-1 Comparação modelos *5 Ultra* e *14 Ultra*

	5DT Data Glove 5 Ultra	***5DT Data Glove 14 Ultra***
Resolução dos sensores	12 bits A/D (alcance típico de 10 bits)	12 bits A/D (alcance típico de 10 bits)
Número de sensores e localização	5 Sensores de flexão baseados em fibra óptica. Um sensor por dedo localizado entre a junta e primeira junção (nas falanges)	14 Sensores de flexão baseados em fibra óptica. 2 Sensores por dedo, um na junta e outro na primeira junção. 1 Sensor por cada zona interdigital.
Ritmo de amostragem	75Hz (mínimo)	75Hz (mínimo)

Para melhor ilustrar a localização dos diversos sensores são mostradas abaixo duas figuras obtidas a partir da documentação oficial, a primeira referente ao modelo de cinco sensores e a segunda em relação ao modelo de catorze sensores.



(a) **(b)**

Figura 2-5 Mapeamento luva
(a) 5 sensores (b) 14 sensores

2.2 Dispositivos imersivos de saída

2.2.1 Óculos Virtuais *eMagin Z800 3D Visor*

Nesta secção é descrito o modo de funcionamento da componente de visualização dos óculos de realidade virtual *eMagin Z800 3D Visor*. Para melhor perceber o seu funcionamento é primeiramente explicado o que é **estereoscopia**, quais os tipos de estereoscopia existentes, e como este conceito se aplica aos óculos utilizados.

Uma das possíveis definições de estereoscopia é, a criação de uma visão tridimensional através da junção de duas vistas de ângulos ligeiramente diferentes sobre uma cena em cada uma das retinas dos olhos. Diferentes técnicas podem ser utilizadas para criar estereoscopia [16], [17]. Alguns exemplos serão agora descritos:

- ***Anaglyph*:** Neste método as imagens são criadas através de duas camadas de cores sobrepostas que contêm uma certa discrepância entre estas para permitir um efeito de profundidade de campo. A imagem principal é composta por duas imagens filtradas através de cores diferentes, uma para cada olho. O objecto principal costuma estar no centro, o cenário em frente e o fundo ligeiramente deslocado. De modo a poder visionar o efeito de

estereoscopia é necessário utilizar óculos nos quais cada lente é constituída por um filtro de cor que corresponda às respectivas camadas utilizadas na imagem. Consultar [18].

Figura 2-6 ***Anaglyph Stereo***

- ***Passive Stereo*****:** É utilizada a polarização da luz, linear ou circular, para transmitir de forma diferenciada as imagens para os respectivos olhos. O par *stereo* é renderizado, dividido pelo *hardware* através da utilização de filtros e a imagem final é posteriormente reconstruída pelos óculos com filtros para o efeito. Consultar [16].

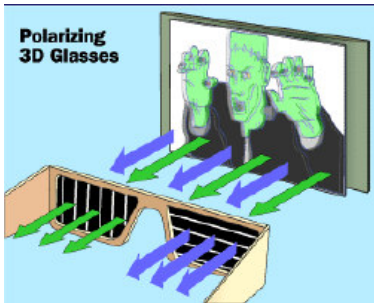


Figura 2-7 ***Passive Stereo***

- ***Active Stereo*****:** O princípio base de estereoscopia activa é o de alternar de forma rápida o envio de duas imagens, cada uma destas para um olho diferente. Os óculos alteram a visualização para o olho direito e esquerdo com as respectivas imagens recebidas. A frequência a que é trocada a imagem entre os olhos costuma ser controlada pelo ***Vertical Sync*** da placa gráfica. Os óculos ***eMagin Z8003D Visor*** encontram-se nesta categoria, pelo que contêm **um *microdisplay* de matriz activa *Organic Light Emitting Diod, OLED-on-silicon*** [12] em cada olho, responsável pela visualização em cada um dos respectivos olhos.

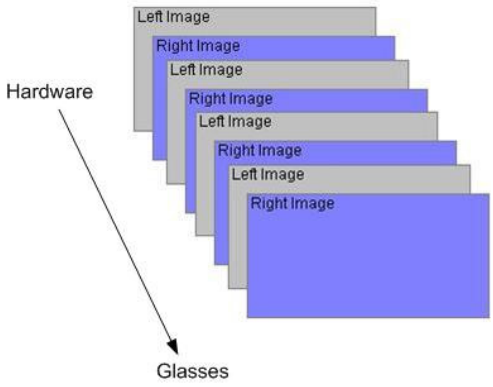


Figura 2-8 ***Active Stereo***

2.3 *Software* de Desenvolvimento

2.3.1 *Dassault Systèmes Virtools*

Para o desenvolvimento das aplicações que façam uso dos referidos dispositivos é utilizado o *software Virtools*.

Virtools é uma plataforma profissional que permite desenvolver, de forma extremamente rápida e intuitiva, conteúdo *3D* interactivo para computadores pessoais e consolas de jogos. Recorre a um paradigma de programação proprietário denominado por ***PCS***, ***Product Context Scenario***, que permite uma programação baseada em **comportamentos** e que não requer, num nível básico, a escrita de uma única linha de código. Pretende-se com o paradigma *PCS* dar a possibilidade de se criarem aplicações recorrendo apenas a operações de ***drag & drop***. No entanto tal só é praticável para aplicações que tenham um nível de complexidade simples.

O *layout* principal da *Virtools* ao ser executada é dado pela figura abaixo, sendo constituído por três áreas principais: ***3D Layout***, ***Buildind Blocks & Data Resources*** e ***Level Manager & Schematic Window***.

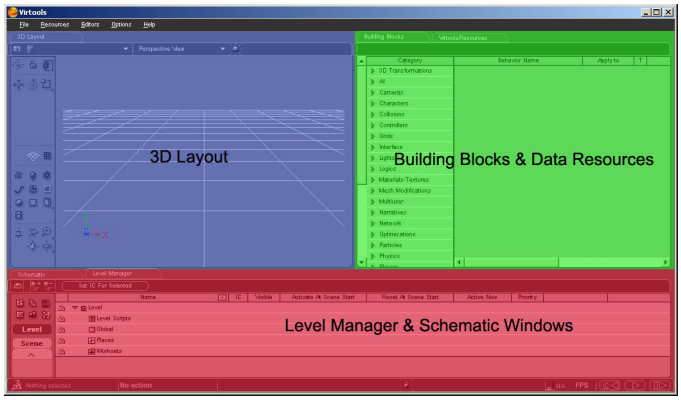


Figura 2-9 ***Layout*** **principal** ***Virtools***

- ***3D Layout*****:** Nesta zona é dada a possibilidade ao utilizador de criar objectos ditos abstractos, comuns num motor de jogo, tais como câmaras ou fontes de iluminação. É também permitido manipular em tempo real o conteúdo 3D ou 2D de uma aplicação através de operações de

translação, rotação ou escala, estas que serão mais detalhadas no capítulo 3. Permite igualmente modificar os parâmetros da câmara seleccionada tal como ***field of view***, **orientação** ou ***panning***.

- ***Building Blocks & Data Resources*:** Aqui estão localizados os diversos comportamentos disponíveis ao utilizador e recursos de dados nativos à *Virtools*, tais como imagens, malhas poligonais, sons, etc.
- ***Level Manager & Schematic Window*:** Permite listar todas as componentes contidas na aplicação e manipular as suas propriedades, criar entidades lógicas e criar e modificar *scripts*. Estes últimos irão conter os comportamentos constituintes da aplicação em desenvolvimento e definir a sequência de processamento desta. Os **comportamentos** estão na forma de entidades denominadas por **b*ehavioural Building Blocks* (*BB*)**, ou **Blocos de Comportamento** e, através destes, é programada a forma como um certo elemento interage com o utilizador e com os restantes elementos existentes na aplicação. Na figura abaixo encontram-se alguns exemplos de blocos de comportamento.

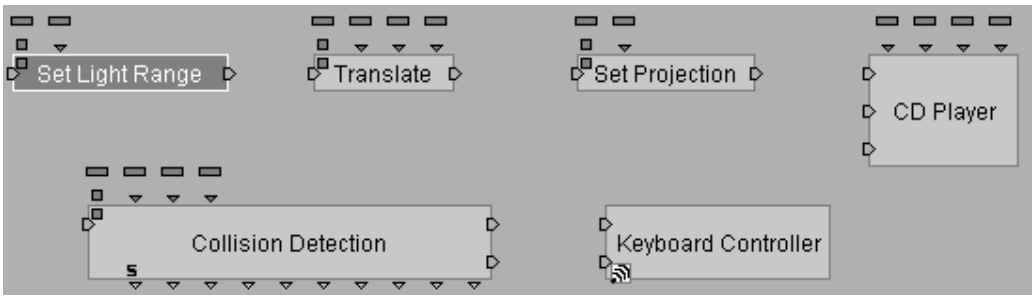


Figura 2-10 Exemplo blocos de comportamentos/*Behavioural Building Blocks*

Um bloco de comportamento para iniciar o seu processamento necessita de receber um sinal de activação, este que é originado por um bloco especial de nome ***Start*** existente em cada *script*.

A constituição básica de um bloco de comportamento é de:

- Pelo menos uma entrada comportamental **(*bIn*)** que conduz o sinal de activação ao bloco de comportamento e é responsável por iniciar o seu processamento
- Zero, uma ou mais saídas comportamentais **(*bOut*)** que são activadas quando o processamento efectuado no bloco de comportamento termina
- O sinal de activação é depois conduzido para novos blocos de comportamento através de ligações comportamentais **(*bLinks*)**.

A passagem de dados entre comportamentos é feita através de:

- Parâmetros de entrada **(*pIn*)** por um **nome**, **tipo** e **valor**
- Parâmetros de saída **(*pOut*)** por um **nome**, **tipo** e **valor**
- Os parâmetros são passados através de ligações de parâmetros **(*pLink*)** ou por atalhos de parâmetros.

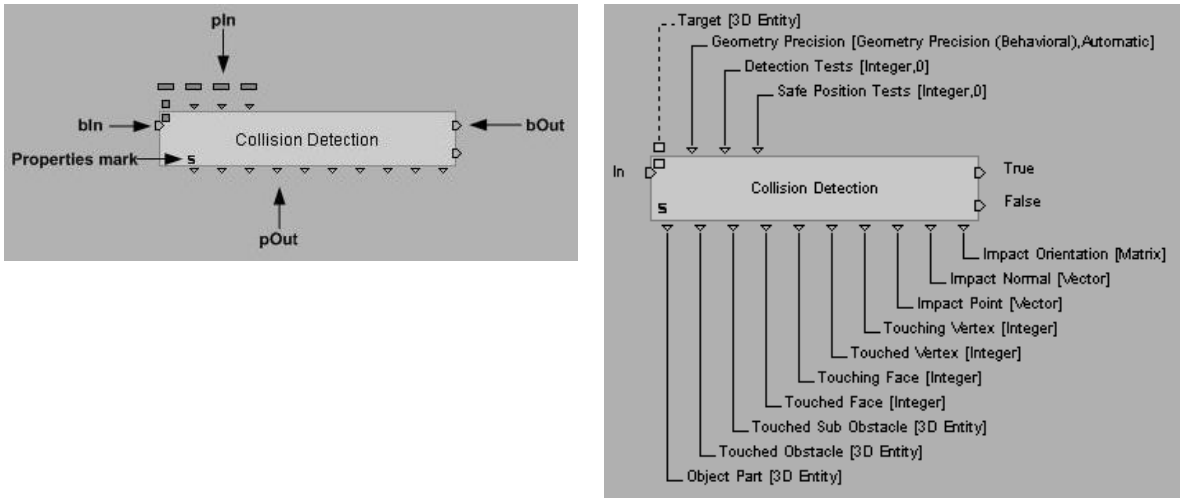


Figura 2-11 Componentes de um bloco de comportamento

Cada bloco de comportamento contém igualmente uma marca (***Properties mark***) que indica que tipo de propriedades que este contém. Esta indica se o bloco de comportamento:

- Contêm caixa de configuração avançada
- Pode enviar/receber mensagens entre *scripts*
- Pode alterar tipo de processamento efectuado sobre os seus parâmetros
- Pode inserir novos ***bIn***, ***bOut***, ***pIn***, ***pOut***

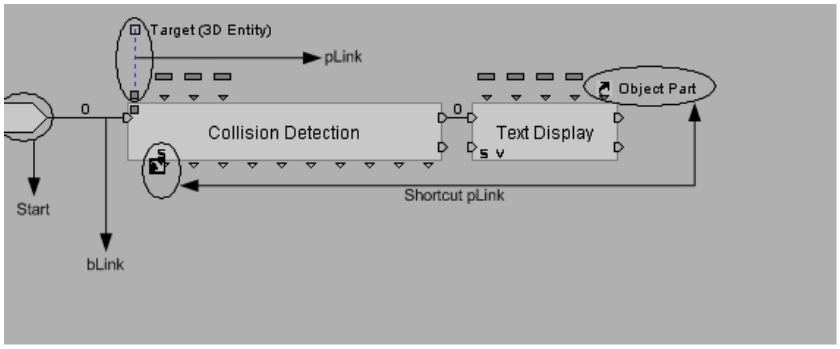


Figura 2-12 Localização de *Start*, *bLink*, *pLink*, *shortcut pLink*

Como exemplo é mostrado na figura 2-13 um bloco de comportamento, ***Collision Detection***. Este contém:

- Uma ***bIn***
- Duas ***bOut***
- Quatro ***pIn***
- Dez ***pOut***.

A função deste bloco de comportamento é de efectuar detecção de colisões entre um objecto 3D presente no cenário e um grupo de obstáculos previamente estabelecidos. Para isso é primeiro necessário activar a ***bIn*** do bloco de comportamento para que este inicie o seu processamento interno. O processo de activação é efectuado, como se pode verificar na figura, através do encaminhamento do sinal de activação desde o elemento de ***Start*** até ao ***bIn*** através de um ***bLink***.

De seguida o bloco de comportamento utiliza informação dos parâmetros de entrada para determinar qual o objecto 3D e grupo de obstáculos a considerar nas colisões e inicia de seguida a detecção de colisões entre estes. Caso não haja colisão, ***bOut False*** é activada indicando que não houve colisão, caso contrário ***bOut True*** é activado indicando colisão e os valores de ***pOut*** são modificados de modo a indicar informação resultante à colisão que está a ocorrer. A informação resultante da colisão pode ser utilizada ligando os seus ***pOut's*** a ***pIn's*** de outros blocos comportamentais como é exemplificado na figura, em que o bloco de comportamento ***Text Display*** imprime para o ecrã o resultado do ***pOut Object Part***.

Nas figuras 2-14 e 2-15 abaixo são exemplificadas operações de ***drag & drop*** tanto para recursos como para blocos de comportamento.

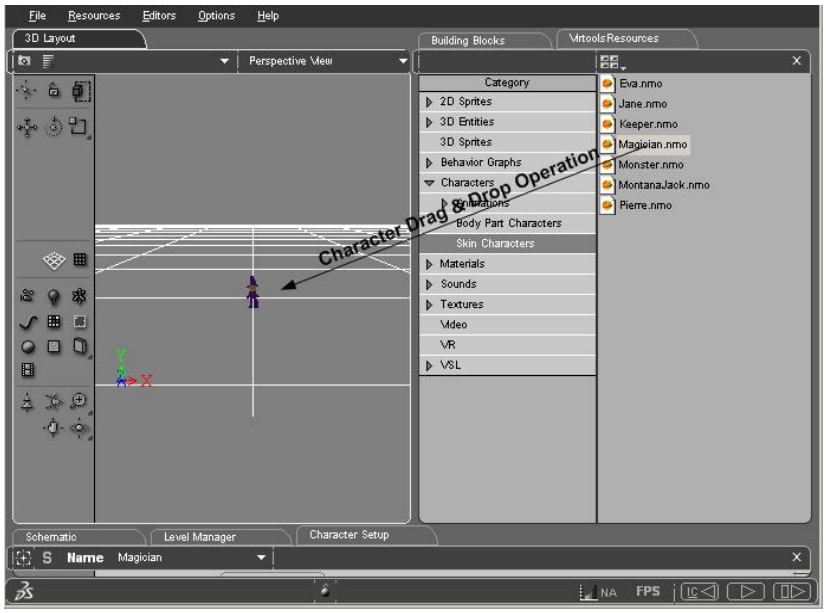


Figura 2-13 ***Drag & Drop*** **de Recursos**

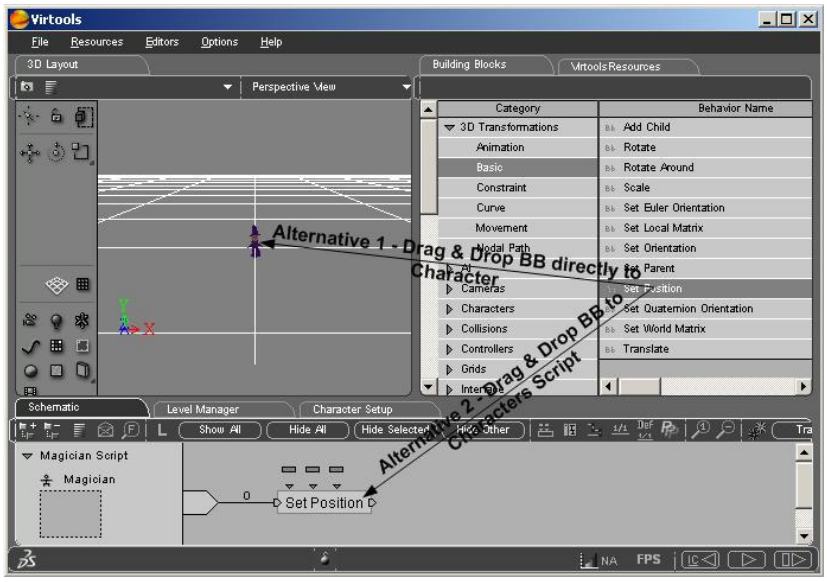


Figura 2-14 ***Drag & Drop*** **de blocos de comportamentos**

A *Virtools* fornece ainda aos seus utilizadores um *SDK*, *Software Development Kit*, para *C++* que permite criar ou modificar blocos de comportamentos e criar novos *plugins*. É igualmente fornecido pela *Virtools* uma linguagem de *scripting*, ***VSL***, ***Virtools Scripting Language***, que, apesar de ter um número menor de funcionalidades em relação ao o *SDK*, permite fazer prototipagem de novos blocos de comportamentos dentro do ambiente *Virtools*.

É apresentada uma lista de acetatos sobre *Virtools* em Anexo – C para informação mais detalhada, desenvolvidos no âmbito de uma apresentação feita para a cadeira de Complementos de Visualização, pertencente à Licenciatura em Engenharia Informática e de Computadores, leccionada pelo professor João Madeiras Pereira.

3 Integração de Dispositivos Imersivos em *Virtools*

A utilização dos diversos dispositivos imersivos não está disponível à partida e de uma forma directa dentro da plataforma *Virtools*, pelo que é necessário desenvolver *plugins* para cada um daqueles, recorrendo ao *SDK* fornecido pela *Virtools* e às suas respectivas *API's, Application Programming Interface*. O presente capítulo tem o objectivo de explicar o processo de integração para cada um dos dispositivos na plataforma *Virtools*.

3.1 *Polhemus Fastrak*

É utilizada a ***API FTApi*** fornecida pela *Polhemus* de modo a estabelecer uma ligação *USB* entre o *PC* e o *tracker* magnético. Após a ligação estar devidamente estabelecida pode ser retirada, para cada receptor ligado ao filtro, informação da sua orientação e posição relativamente ao emissor, através de três ângulos de ***Euler***, ***Azimuth***, ***Elevation*** e ***Roll*** e coordenadas cartesianas **(*x,y,z*)**.

Numa primeira abordagem, é desenvolvido um bloco de comportamento responsável pelo controlo total do *tracker*, constituída por um parâmetro de entrada que indique a estação/receptor à qual se quer retirar informação e fornece como parâmetros de saída a respectiva posição e orientação.

No entanto, com o bloco de comportamento a ser responsável pela comunicação com o *tracker*, baixa em muito o desempenho da aplicação limitando o número de fps, *frames* por segundo de **60** a aproximadamente **5-10**. De modo a ultrapassar esta limitação é criada, para além do bloco de comportamento, um Gestor, uma entidade especial da *Virtools*, que tal como o nome indica, gere recursos de dados em *background* na altura da execução de aplicações. O Gestor é responsável pela comunicação com o *tracker* e faz a ponte deste com o bloco de comportamento a utilizar nas aplicações. Com este mecanismo consegue-se obter uma melhoria para cerca de **30** ***fps*** nas aplicações que façam uso do dispositivo. O inconveniente da utilização de um Gestor é de que é necessário ligar o *tracker* antes de se executar a aplicação.

Na figura abaixo encontra-se o aspecto visual final do bloco de comportamento criado para o *Polhemus Fastrak*.

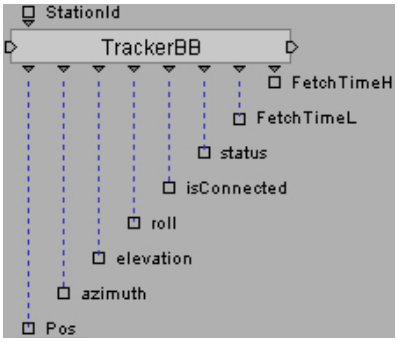


Figura 3-1 Bloco de comportamento *TrackerBB*

O comportamento contém:

- Parâmetro de entrada de tipo **Inteiro** com nome ***StationId*** que permite seleccionar qual das quatro estações receptoras disponíveis se quer retirar dados
- Parâmetro de saída do tipo **vector de floats** de nome ***Pos*** com a informação espacial do receptor identificado por ***StationId***,
- Parâmetro de saída do tipo ***float*** de nome ***azimuth***, indica em graus o **ângulo de Euler *azimuth*** do receptor ***StationId***
- Parâmetro de saída do tipo ***float*** de nome ***elevation***, indica em graus o **ângulo de Euler *elevation*** do receptor ***StationId***
- Parâmetro de saída do tipo ***float*** de nome ***roll***, indica em graus o **ângulo de Euler *roll*** do receptor ***StationId***
- Parâmetro de saída do tipo ***String*** de nome ***isConnected***, indica sucesso da ligação
- Parâmetro de saída do tipo ***String*** e nome ***status*** indica informação de *debug*
- Parâmetro de saída do tipo **Inteiro** de nome **FetchTimeL** indica 32 bits mais à esquerda do tempo de latência interno, em **µs**, do Gestor
- Parâmetro de saída do tipo Inteiro de nome **FetchTimeH** indica 32 bits mais à direita do tempo de latência interno, em **µs**, do Gestor

O gestor, antes de cada *frame* ser processada, comunica com o *tracker* de modo a obter os dados necessários. Não é feito nenhum teste de se o *tracker* é desligado após se fazer a ligação inicial, isso porque a referida operação provoca um atraso no desempenho global do sistema Gestor+*BB* e assim para termos de optimização foi retirado.

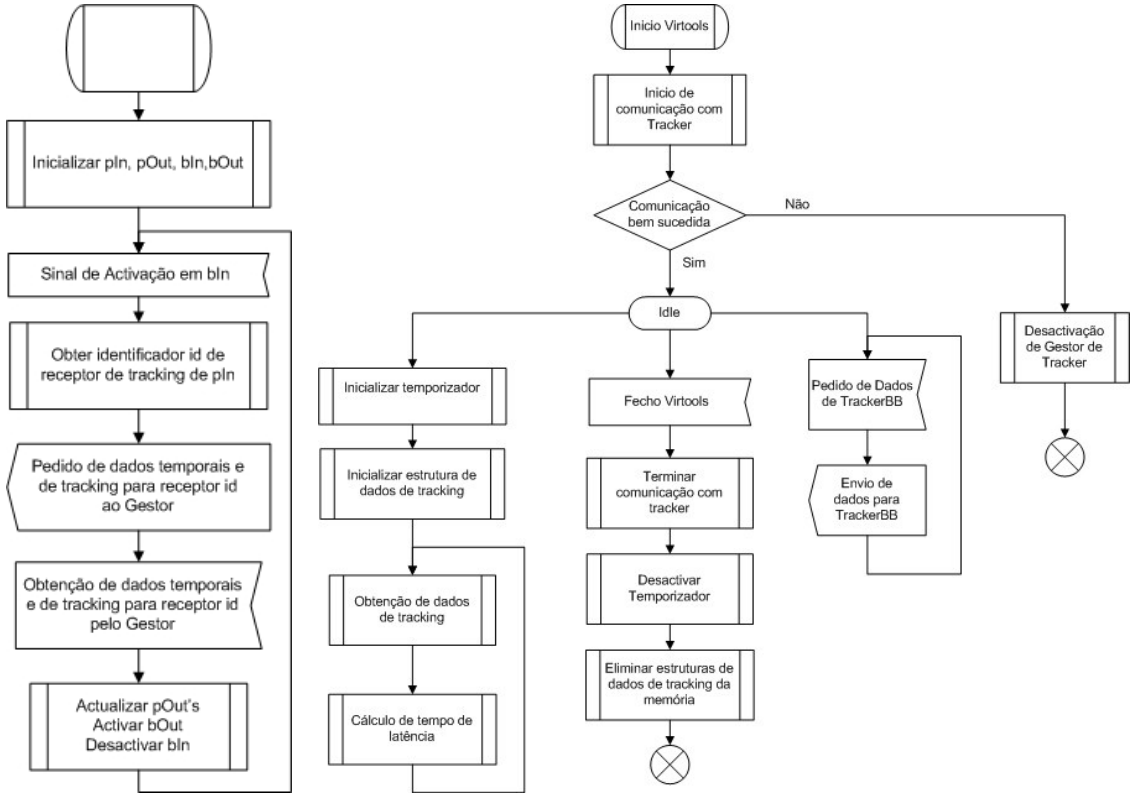
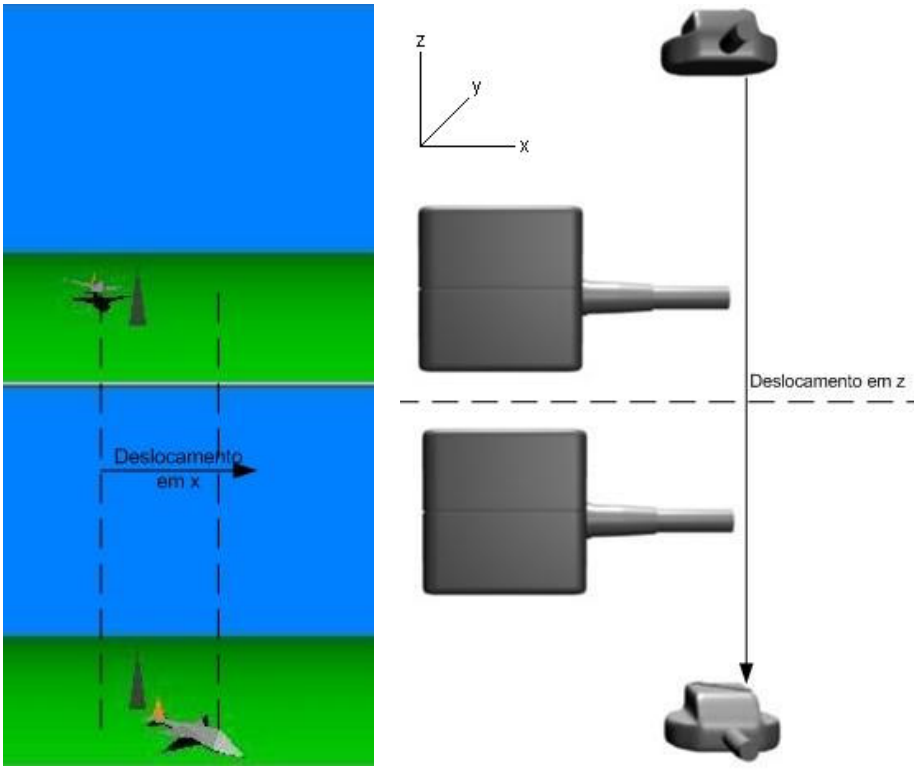

Inicializar pIn, pOut, bIn,bOut
Sinal de Activação em bIn
Obter identificador id de receptor de tracking de pIn
Pedido de dados temporais e de tracking para receptor id ao Gestor
Obtenção de dados temporais e de tracking para receptor id pelo Gestor
Actualizar pOut's
Activar bOut
Desactivar bIn
Inicio Virtools
Inicio de comunicação com Tracker
Comunicação bem sucedida
Não
Sim
Idle
Desactivação de Gestor de Tracker
Inicializar temporizador
Inicializar estrutura de dados de tracking
Obtenção de dados de tracking
Cálculo de tempo de latência
Fecho Virtools
Terminar comunicação com tracker
Desactivar Temporizador
Eliminar estruturas de dados de tracking da memória
Pedido de Dados de TrackerBB
Envio de dados para TrackerBB

Figura 3-2 Funcionamento *TrackerBB* (à esquerda) + Gestor (à direita)

Para a utilização do comportamento é necessário referir algumas considerações:

- É fundamental ter sempre em conta que os valores de coordenadas dados pelo receptor são sempre em relação à posição e orientação do emissor logo, é necessário **definir à *priori* uma posição e orientação por defeito do emissor** de modo a que poder atribuir uma interpretação fixa aos dados obtidos.
- Devido à natureza magnética do *tracker*, é necessário **escolher uma área de movimentação do receptor** dado que este tem comportamentos diferentes dependentes da zona de utilização, *p.e.* imaginando uma linha de equador no emissor, caso se passe essa linha os valores dados passam a ter interpretações diferentes. A figura abaixo exemplifica o que foi anteriormente dito, a segunda imagem é obtida fazendo uma deslocação apenas no eixo Z do receptor. Como se pode verificar com um deslocamento apenas no eixo Z altera as coordenadas z como também as coordenadas x do objecto.



(a) Deslocamento virtual **(b) Deslocamento real**

Figura 3-3 Má utilização espacial do tracker magnético

- Por último, a **utilização dos ângulos de *Euler* deve ser aplicada pela ordem *azimuth*, *elevation* e finalmente *roll***, isto porque a aplicação de rotações não é comutativa (multiplicação de matrizes [de rotação] não é comutativa). Caso não se respeite a ordem dada pela *Polhemus* poderá haver inconsistências na orientação final do objecto 3D.

3.2 Luvas *5DT*

De modo a integrar a informação dada pelas luvas no ambiente *Virtools* é utilizada a ***API FGlove*** da *5DT*. Mais uma vez a ligação é estabelecida através das portas *USB*. Caso se ligue apenas uma luva, independentemente de ser mão esquerda ou direita, é-lhe automaticamente atribuído o endereço *USB0*. No caso de se ligar um par de luvas, o endereço *USB0* é sempre atribuído à luva da mão direita e o endereço *USB1* é atribuído à luva da mão esquerda. Tal como para o *tracker*, é utilizado um Gestor que efectua a comunicação entre o bloco de comportamento a ser criado e as luvas. O Gestor, no momento de execução da *Virtools*, detecta o número de luvas conectadas às portas *USB* e cria uma ligação específica para cada uma delas. Mais uma vez para fins de optimização, tal como foi feito para o *tracker*, na altura da recolha de dados não é efectuado a verificação de se ter desligado a luva após o gestor ter criado uma ligação.

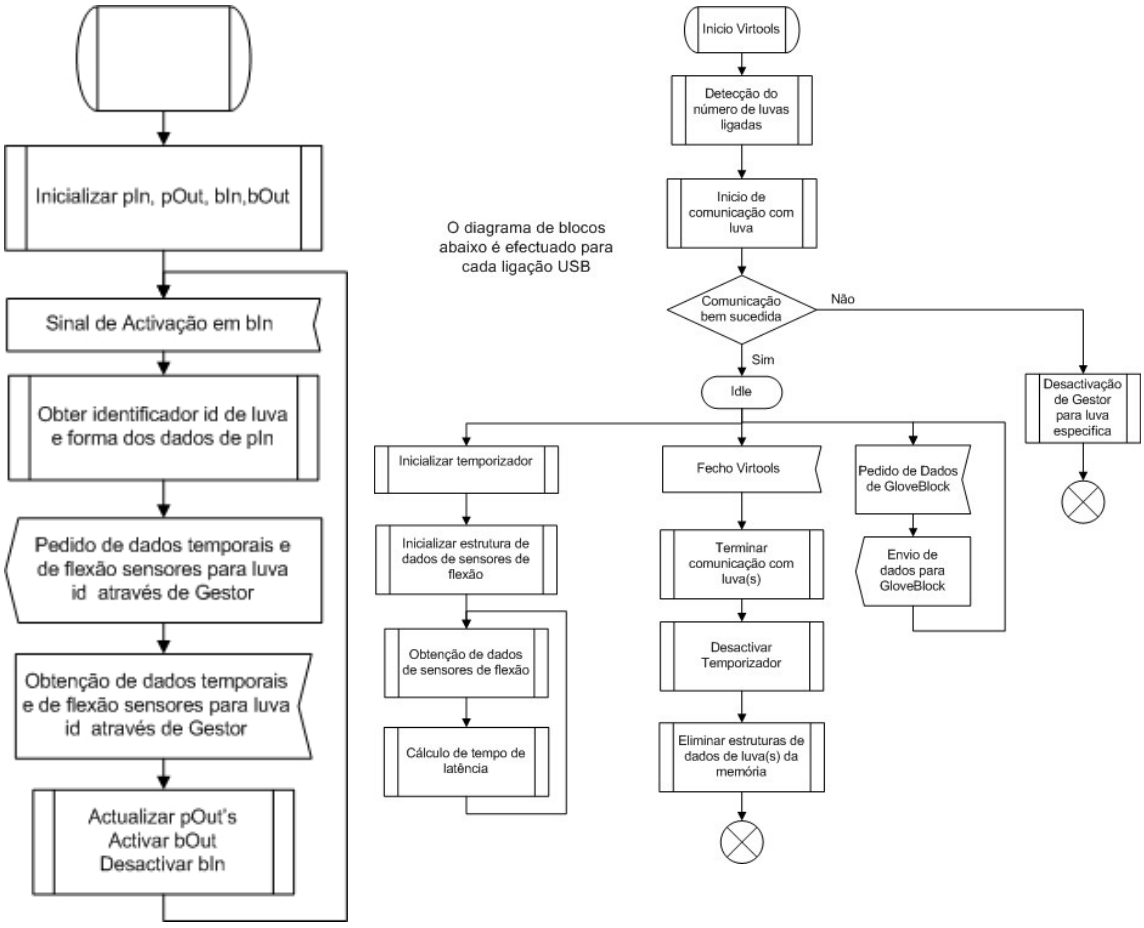

Inicializar pIn, pOut, bIn,bOut
Sinal de Activação em bIn
Obter identificador id de luva e forma dos dados de pIn
Pedido de dados temporais e de flexão sensores para luva id através de Gestor
Obtenção de dados temporais e de flexão sensores para luva id através de Gestor
Actualizar pOut's
Activar bOut
Desactivar bIn
O diagrama de blocos abaixo é efectuado para cada ligação USB
Inicio Virtools
Detecção do número de luvas ligadas
Inicio de comunicação com luva
Comunicação bem sucedida
Não
Sim
Idle
Desactivação de Gestor para luva especifica
Inicializar temporizador
Inicializar estrutura de dados de sensores de flexão
Obtenção de dados de sensores de flexão
Cálculo de tempo de latência
Fecho Virtools
Terminar comunicação com luva(s)
Desactivar Temporizador
Eliminar estruturas de dados de luva(s) da memória
Pedido de Dados de GloveBlock
Envio de dados para GloveBlock

Figura 3-4 Funcionamento de *GloveBlock* (à esquerda) + Gestor (à direita)

É necessário ter em conta igualmente o número de sensores que a luva contém, no entanto para fins de optimização o mesmo tipo de operações é efectuada independente do tipo de luva ligada, pelo que a *API* da *5DT* retorna automaticamente o valor 0 caso o sensor ao qual se tenha pedido não esteja mapeado na luva em questão. O comportamento criado para obter os dados do Gestor das luvas fica com o seguinte aspecto final.

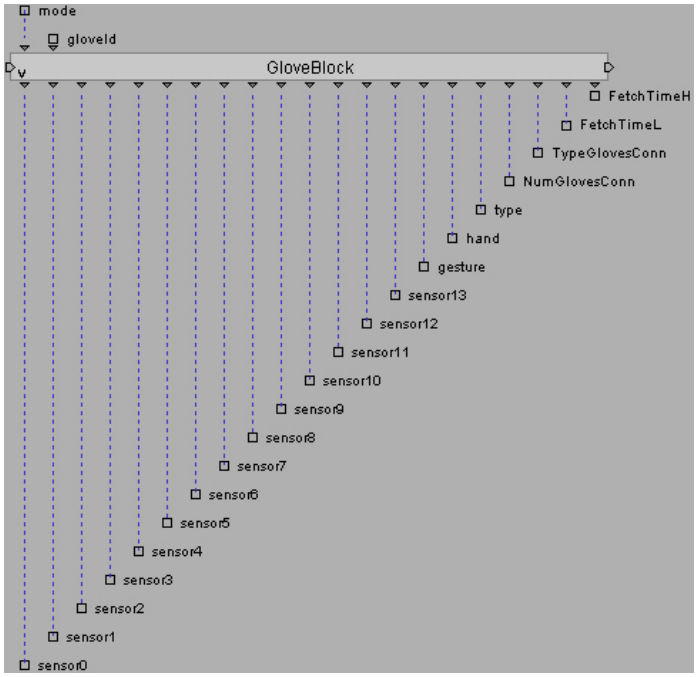


Figura 3-5 Bloco de comportamento *GloveBlock*

Os parâmetros deste bloco de comportamento indicam:

- Parâmetro de entrada de tipo **Inteiro** de nome ***mode***, que refere o modo pelo qual são obtidos os dados. Os modos disponíveis são ***raw*** e ***auto-calibrated***, em que no primeiro os dados são apresentados tal e qual como são recebidos pela luva, com um valor entre 0 e 4095, pelo que 0 é nenhuma flexão e 4095 é flexão total do sensor, e no segundo são auto-calibrados de modo a ficarem com um valor entre 0 e 1, em que 0 é nenhuma flexão e 1 é flexão total do sensor.
- Parâmetro de entrada de tipo **Inteiro** de nome **gloveId** que deverá ser colocado do porto *USB* da luva para o qual se quer receber a informação sensorial.
- Parâmetros de saída [0,13] do tipo **float** de nome **sensor[0,13]** indicam o valor de flexão de cada um dos sensores existentes. No caso da luva de 14 sensores, os parâmetros têm correspondência directa com o sensor a que estão associados, ver figura 2-4. Para a luva de 5 sensores o mapeamento entre os parâmetros do comportamento e os sensores da luva são os especificados pela tabela abaixo e identificados na figura 2-3. Aos parâmetros com identificador 2,5,8 e 11 é retornado o valor 0 dado que não há nenhuma correspondência entre estes e os sensores disponíveis.

Tabela 3-1 Identificação de sensores activos

Sensor	Identificador do Sensor
A	0,1
B	3,4
C	6,7
D	9,10
E	12,13

- Parâmetro de saída do tipo **Inteiro** de nome ***gesture***, indica qual é o gesto que a mão está a fazer de entre um total de 16 gestos *standard*. O valor 16 é determinado pelo facto de a API internamente fazer o processamento de gestos recorrendo apenas à informação de 4 dedos, tendo assim acesso a $2^4 = 16$ combinações possíveis. O mapeamento dos gestos é o indicado pela figura abaixo.

0) Fist
1) Index finger point
2) Middle finger point
3) Two finger point
4) Ring finger point
5) Ring-index finger point
6) Ring-middle finger point
7) Three finger point (or not little finger point)
8) Little finger point
9) Index and little finger point
10) Little-middle finger point
11) Not ring finger point
12) Little-ring finger point
13) Not middle finger point
14) Not index finger point
15) Flat hand

Figura 3-6 Mapeamento de gestos

- Parâmetro de saída do tipo ***String*** de nome **hand**, indica tipo de luva que está a ser processada pelo comportamento
- Parâmetro de saída do tipo ***String*** de nome **type**, indica se luva é de a mão direita ou esquerda
- Parâmetro de saída do tipo ***String*** de nome **numGlovesConn**, indica número total de luvas ligadas ao Gestor
- Parâmetro de saída do tipo ***String*** de nome **typeGlovesConn**, indica número de sensores disponíveis.
- Parâmetro de saída do tipo **Inteiro** de nome **FetchTimeL** indica 32 bits mais à esquerda do tempo de latência interno, em µs, do Gestor
- Parâmetro de saída do tipo **Inteiro** de nome **FetchTimeH** indica 32 bits mais à direita do tempo de latência interno, em µs, do Gestor

3.3 Óculos virtuais *eMagin Z800*

A integração dos óculos em *Virtools* é composta por duas partes:

- **Seguimento**
- **Visualização**

3.3.1 Seguimento

A integração da componente de seguimento é importante pelo que desta forma poder-se-á saber quais os movimentos que estão a ser feitos pela cabeça do utilizador dos óculos e utilizar convenientemente essa informação.

É desenvolvido de forma análoga aos dispositivos anteriores um Gestor de dispositivos e um bloco de comportamento de nome ***eMaginBB***. O gestor é responsável pela comunicação com os óculos através do ***EMA SDK v2.2*** enviando dados para o bloco de comportamento sempre que solicitado.

Através do *SDK* acima especificado não é possível obter os **ângulos de *Euler*** directamente do giroscópio interno dos óculos. È definida uma função de ***CALLBACK*** que é executada/chamada pela API com uma periodicidade dada por um certo **tempo de *polling*** e é através desta que são obtidos os dados necessários. O valor por defeito do **tempo de *polling*** é de **100ms**.

O bloco de comportamento tem o seguinte aspecto.

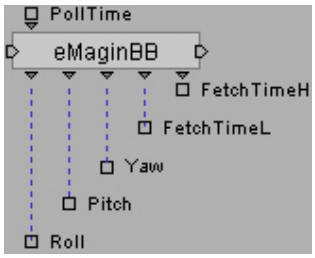


Figura 3-7 Bloco de comportamento *eMaginBB*

Apenas contém um parâmetro de entrada:

- ***PollTime***, do tipo ***float***. Permite ao utilizador modificar o tempo de ***polling***. O valor colocado tem com unidades ***ms*** e deverá estar contido no intervalo **[*30, 60000*]**

Contém os seguintes parâmetros de saída:

- ***Roll***, do tipo ***float***, indica em graus o **ângulo de *Roll*** actual dos óculos
- ***Pitch***, do tipo ***float***, indica em graus o **ângulo de *Pitch*** actual dos óculos
- ***Yaw***, do tipo ***float***, indica em graus o **ângulo de *Yaw*** actual dos óculos
- ***FetchTimeL***, do tipo **Inteiro**, indica 32 bits mais à esquerda do tempo de latência interno, em **µs**, do Gestor
- ***FetchTimeH***, do tipo **Inteiro**, indica 32 bits mais à direita do tempo de latência interno, em **µs**, do Gestor

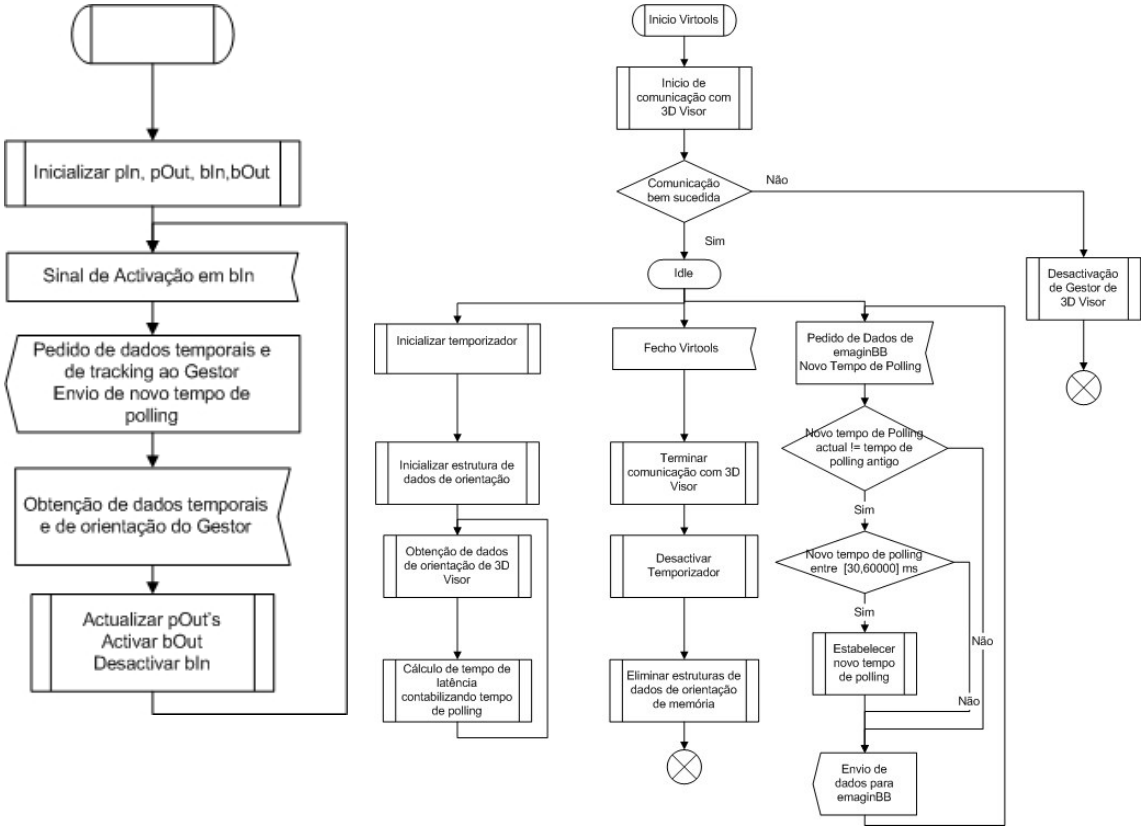

Inicializar pIn, pOut, bIn,bOut
Sinal de Activação em bIn
Pedido de dados temporais e de tracking ao Gestor
Envio de novo tempo de polling
Obtenção de dados temporais e de orientação do Gestor
Actualizar pOut's
Activar bOut
Desactivar bIn
Inicio Virtools
Inicio de comunicação com 3D Visor
Comunicação bem sucedida
Não
Sim
Idle
Desactivação de Gestor de 3D Visor
Inicializar temporizador
Fecho Virtools
Pedido de Dados de emaginBB
Novo Tempo de Polling
Inicializar estrutura de dados de orientação
Terminar comunicação com 3D Visor
Novo tempo de Polling actual != tempo de polling antigo
Sim
Obtenção de dados de orientação de 3D Visor
Desactivar Temporizador
Novo tempo de polling entre [30,60000] ms
Sim
Não
Cálculo de tempo de latência contabilizando tempo de polling
Eliminar estruturas de dados de orientação de memória
Estabelecer novo tempo de polling
Não
Envio de dados para emaginBB

Figura 3-8 Funcionamento de *eMaginBB* (à esquerda) + Gestor (à direita)

3.3.2 Visualização

Para a integração da componente de visualização dos óculos VR com a plataforma *Virtools* é utilizado o *plugin* oficial de realidade virtual ***VR PACK***. No entanto, antes de ser explicada a integração dos óculos com a *Virtools* é necessário introduzir noções de boa definição e utilização de estereoscopia. Primeiro é apresentado o conceito de ***Projection Referencial Cameras*** e o porquê da sua utilização em detrimento de uma câmara convencional.

Uma **câmara convencional** em *Virtools* é definida por um ***frustum* simétrico** através de:

- **Posição**
- **Orientação**
- ***Field of view***
- ***Aspect ratio***
- ***Near* e *far clips***

Ao contrário das ***Projection Referencial Cameras*** que são definidas por um ***frustum* assimétrico** através de:

- **Posição**
- ***3D Sprite***
- **N*ear* e *far clips*.**

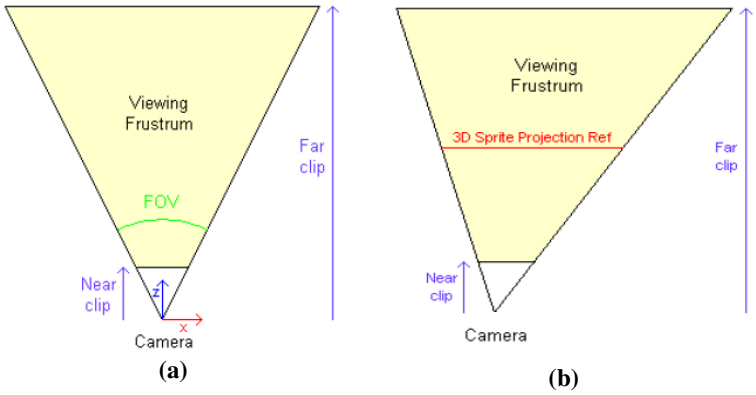


Figura 3-9 Definição de câmeras
(a) Câmara normal (b) *Projection Referential Camera*

Pelo que uma ***3D Sprite***, em *Virtools*, é um quadrado/imagem 2D com uma certa posição e orientação pelo que pode ter a sua orientação restrita a um certo eixo ou livre.

Um esquema correcto de estereoscopia tem o seguinte aspecto:

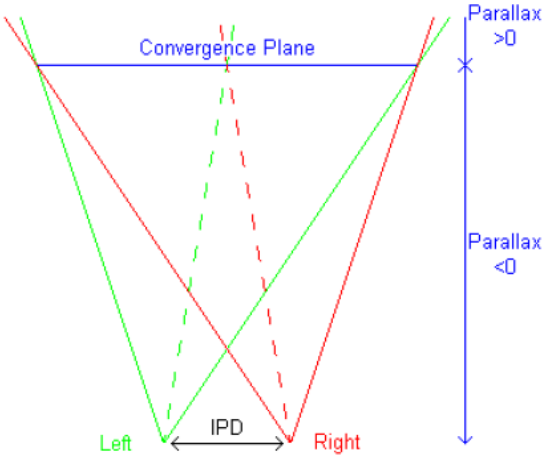


Figura 3-10 Esquema correcto de estereoscopia

- ***Convergence Plane****:* Tal como é indicado pelo próprio nome, especifica o plano de convergência em que as imagens geradas por ambas as câmaras são idênticas.
- ***Parallax***: Representa a diferença entre a imagem direita e esquerda de um certo objecto. Objectos com um *parallax* negativo são percepcionados antes do plano de convergência, com um *parallax* positivo são percepcionados atrás do plano de convergência e com um *parallax* nulo são percepcionados no plano de convergência.
- ***IPD*** (***InterPupillar Distance***): Este valor indica a distância entre as duas câmaras que, deve ser igual à distância entre o centro das pupilas humanas.

Com um bom esquema de estereoscopia existem zonas de paralax positivo, negativo e zero, tendo assim objectos a serem percepcionados antes, coincidente e depois do plano de convergência ou visualização.

A razão pela qual as ***Projection Referential Cameras*** são utilizadas em detrimento das câmaras convencionais é mostrada através das figuras 3-11 e 3-12. Com uma câmara convencional o esquema de estereoscopia é o seguinte:

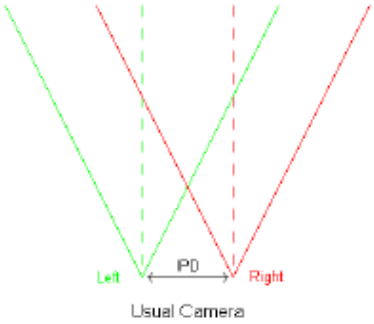


Figura 3-11 Estereoscopia com câmara convencional

Como se pode notar, com as duas câmaras a olharem na mesma direcção os **objectos têm sempre um *parallax* negativo** e o **plano de convergência localizado em infinito**, sendo os objectos sempre percepcionados em frente ao plano de convergência o que, claramente contradiz a visão percepcionada na vida real. Já com a utilização de ***Projection Referential Cameras*** obtém-se o seguinte esquema:

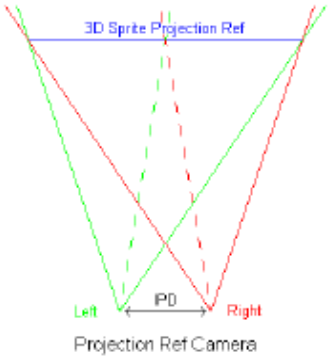


Figura 3-12 Estereoscopia com *Projection Referential Cameras*

É verificado que, com esta configuração se consegue uma correcta definição de estereoscopia, em que a ***3D Sprite*** representa o referencial de projecção e plano de convergência.

Após uma correcta construção da câmara de jogo é necessário integrar o *plugin VR Pack* na aplicação. Uma composição que faça uso do *VR Pack* necessita de ficheiros de configuração externos, de modo a possibilitar diversas configurações de *Hardware* sem que seja necessário a alteração da aplicação. Os ficheiros de configuração são constituídos por ***tokens***, **palavras-chave**, seguidos dos respectivos valores, que poderão ser inteiros, *floats* ou *strings*. Só são abordados os *tokens* necessários para a configuração da estereoscopia, sendo os restantes descritos na documentação oficial da *VR Pack*.

Quando se executa uma aplicação é necessário ter criado um ficheiro com nome *"**VRPack.cfg**"* no mesmo directório desta, pelo que este ficheiro deverá conter ou referenciar toda a informação necessária para a configuração do *VR Pack*. É possível referenciar ficheiros externos dentro de um ficheiro de configuração podendo assim criar uma estrutura modular de ficheiros de configuração.

Apenas serão colocados e explicados os ficheiros de configuração que contenham informação essencial para a definição de estereoscopia.

```
driverId 0

bitPerPixel 32

View_0_0 quad - 0 0 800 600
```

Figura 3-13 Conteúdo de *VRDisplay.cfg*

Dentro do ficheiro ***VRDisplay.cfg***, que é referenciado por ***VRPack.cfg***, só é necessário descrever a linha:

View_0_0 quad - 0 0 800 600

Esta efectua a configuração de estereoscopia activa.

A sintaxe do *token* é a seguinte:

View_hostId_chanId stereomode cameraname x y w h

Sendo que:

- ***hostId***: Lista de índices de *Hosts* começando com o valor zero que indica o *Master*
- ***chanId***: Índice para a vista de determinado *Host*
- ***stereomode***: Indica tipo de ***stereo*** a utilizar
  - ***none***: Não utiliza *stereo*
  - ***quad***/***both***: Utiliza *stereo* activo
  - ***left***: Olho esquerdo em *stereo* passivo
  - ***right***: Olho direito em *stereo* passivo
- ***cameraname***: Nome da câmara tal e qual como aparece na composição, que irá sobrepor a câmara activa. Caso se queira utilizar câmara activa por defeito utilizar parâmetro **-**
- **x y w h:** posição e tamanho da *viewport*

```
StereoInvertEyes 0

InterpupillarDistance 0

FocalLength 1
```

Figura 3-14 Conteúdo de *VRStereo.cfg*

VRPack.cfg referencia igualmente o ficheiro ***VRStereo.cfg.*** Dentro deste, o primeiro *token* indica caso queira-se as imagens estereoscópicas invertidas (*left – right → right – left*) com o valor 1, invertido, e 0, normal. O segundo e terceiro *tokens* são exemplos de ***User Tokens***, isto é, ***tokens* definidos pelo utilizador**. Neste caso os valores de ***InterpuppilarDistance*** e ***FocalLength*** são utilizados para definir, dentro da aplicação, a ***IPD*** e **distância focal** no bloco de comportamento ***VR Set Stereo Settings*** responsável pela configuração de estereoscopia.

Note-se que há limitações impostas ao jogo devido à utilização dos óculos. É necessário limitar a ***frame rate*** do jogo a um máximo de ***30 fps*** e este tem de ser visualizado em modo ecrã inteiro com uma resolução de ***800x600*** a uma taxa de refrescamento de ***60Hz***.

3.3.2.1 Criação de imagens estereoscópicas

Apesar da estereoscopia ser automaticamente gerada pela *Virtools*, é dada uma breve descrição de como é feita a criação das duas imagens estereoscópicas.

Assume-se que:

- Tipo de projecção é perspectiva, ou seja, objectos mais distantes da câmara diminuem o seu tamanho relativo
- Sistema de coordenadas dado pela regra da mão direita
- Plano do ecrã de visualização encontra-se em plano ***xy***
- Ponto genérico ***A*** tem coordenadas ***(x,y,z)***
- Câmara inicialmente situa-se nas coordenadas ***(0,0,d)***

A projecção do ponto genérico **A** no plano de visualização tem como coordenadas:

$$\left(\frac{x\times d}{d-z},\frac{y\times d}{d-z}\right) \tag{3.1}$$

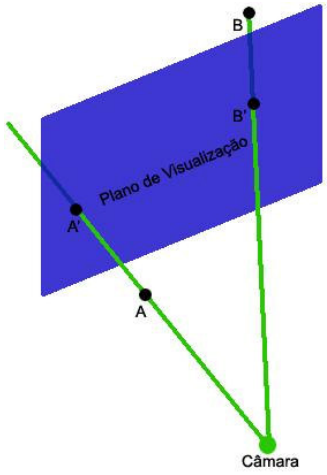


Figura 3-15 Projecção de pontos no plano de visualização

Considerando o seguinte exemplo, a câmara está colocada em (0,0,10) e o ponto A tem como coordenadas (7,3,5). Este vai ser projectado em (14,6) numa câmara de projecção perspectiva convencional.

É agora introduzida mais uma câmara e efectuado um deslocamento a ambas de modo a ficarem com um distância ***c*** entre elas. Para a câmara da esquerda é feito um deslocamento à esquerda de metade da separação total entre câmaras e para a câmara à direita é feito um deslocamento de metade da separação total entre câmaras para a direita.

Nas equações abaixo, em vez de se fazer um deslocamento às câmaras faz-se um deslocamento simétrico ao cenário pelo que esta operação é equivalente à descrita acima.

$$\left(\frac{(x+\frac{c}{2})\times d}{d-z},\frac{y\times d}{d-z}\right), \text{ Para câmara esquerda} \tag{3.2}$$

$$\left(\frac{(x-\frac{c}{2})\times d}{d-z}, \frac{y\times d}{d-z} \right), \text{ Para câmara direita} \quad (3.3)$$

Utilizando novamente o exemplo anterior considerando agora ***A*** em (7, 3, -5). Caso tenha-se uma separação de câmara de uma unidade a câmara esquerda projecta o ponto ***A*** em (5,2) e a câmara direita projecta o ponto em (4,3(3), 2). O ponto ***A*** fica projectado para a direita 0.667 unidades para a direita na câmara da esquerda em relação à câmara da direita, ou seja, a diferença entre a projecção do ponto na câmara direita e na câmara esquerda é negativa ficando assim com um ***paralax* negativo**. Assim este é sempre percepcionado em frente à superfície de visualização. Caso o ponto estivesse deslocado mais à esquerda na projecção da câmara da esquerda em relação à projecção da câmara da direita, então este teria um ***paralax* positivo** e seria percepcionado atrás do plano de visualização Um ***paralax* zero** acontece no caso do ponto ter uma projecção idêntica em ambas as câmaras

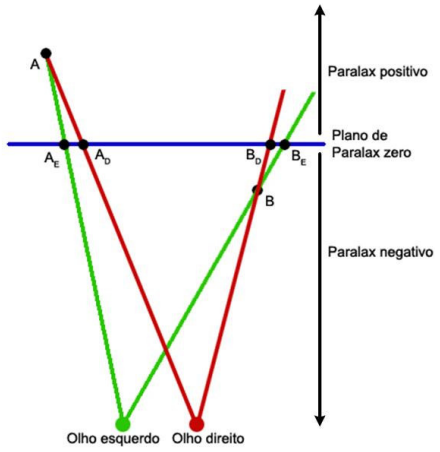


Figura 3-16 Tipos de *paralax*

Imagem estereoscópicas balanceada utilizam tanto *paralax* negativo como positivo, e alguma parte do cenário será projectada no ou próximo do plano de visualização. No entanto utilizando as equações (4.2) e (4.3) faz com que todos os pontos de um cenário sejam projectados com um *paralax* negativo tornando a experiência estereoscópica desagradável à visão do utilizador. Isto pode ser alterado simplesmente deslocando os valores projectados da câmara esquerda para a esquerda e os valores da câmara direita para a direita. Se o deslocamento aplicado aos pontos é igual ao deslocamento original da câmara, a geometria resultante irá colocar o plano de projecção original em *paralax* zero. Os objectos que estejam originalmente em frente ao plano de visualização ficam com um *paralax* negativo e os objectos que estejam atrás do plano de visualização ficam com um *paralax* positivo.

As novas equações ficam assim com o seguinte aspecto:

$$\left(\frac{(x+\frac{c}{2})\times d}{d-z} - \frac{c}{2}, \frac{y\times d}{d-z} \right), \text{ Para câmara esquerda} \quad (3.4)$$

$$\left(\frac{(x-\frac{c}{2})\times d}{d-z} + \frac{c}{2}, \frac{y\times d}{d-z} \right), \text{ Para câmara direita} \quad (3.5)$$

Retomando o exemplo aqui mostrado, o ponto **A** terá agora novas projecções. Este é projectado em (4.5,2) para a câmara esquerda e (4.8(3),2) para a câmara direita, tornando assim o paralax do ponto positivo, fazendo agora sentido dado que a coordenada z original é -5, colocando-o atrás do plano de visualização, este que tem coordenada z igual a zero.

As duas projecções derivadas das últimas equações são chamadas de "***Parallel Axis Assymetric Frustum Perspective Projections***". Note-se que transladando o cenário em sentidos opostos de ***x*** tanto os eixos de projecção, ou vectores de câmara alvo, mantêm-se paralelos. O passo final no qual se desloca o cenário torna os *frustums* de projecção assimétricos, ou seja, para cada projecção é mostrado mais de um cenário num lado do eixo que no outro. Com esse resultado consegue-se obter um par de projecções perspectivas com *frustum* assimétrico balanceando confortavelmente o efeito de *paralax*.

4 Conceitos básicos para a criação de jogos em *Virtools*

Pretende-se com este capítulo apresentar conceitos básicos necessários para a criação de jogos dentro da plataforma *Virtools*.

4.1 Criação e controlo de câmara

A visualização tradicional do conteúdo de jogos em *Virtools* é efectuada através de entidades denominadas ***cameras***, estas que tal como o nome indica utilizam a analogia a uma câmara fotográfica. Uma *camera* em *Virtools* contém:

- **Posição:** Posição da câmara em coordenadas **[x,y,z]** do mundo
- **Orientação:** Orientação da câmara através dos ângulos aplicados a cada um dos seus eixos **[x,y,z]** local
- **Near clip** e **Far clips:** Plano de cortes que ajudam a definir qual a zona do espaço 3D que é visualizada
- **Tipo de projecção:** O tipo de projecção indica qual o volume de visualização pretendido. A *camera* pode ter um tipo de projecção **Perspectiva** ou **Ortográfica**
    - **Projecção Ortográfica**. Cria um volume de visualização na forma de um paralelepípedo rectangular, assim e como o seu tamanho não varia entre os extremos, faz com que os objectos desenhados mantenham sempre o mesmo tamanho independentemente da distância que se encontra do nosso ponto de visualização. As dimensões do volume de visualização são dadas pela especificação de seis planos de corte, **left**, **right**, **top**, **bottom**, **near**, **far**.

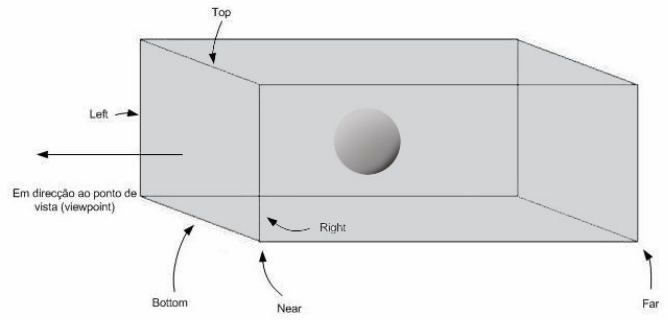


Figura 4-1 Projecção Ortográfica

    - **Projecção Perspectiva:** Com a projecção Perspectiva o volume de visualização tem a forma de um ***frustum*,** pirâmide com o topo cortado. Este tipo de projecção aproxima-se mais da realidade dado que à medida que um objecto se vai afastando do ponto de visualização vai diminuindo as suas dimensões. Isto porque a dimensão

relativa do objecto diminui dado que o volume do *frustum* aumenta. O *frustum* é definido por dois planos de corte, ***Near Clip*** e ***Far Clip***, relação de aspecto entre largura e altura da janela de visualização e abertura da câmara (*field of view*).

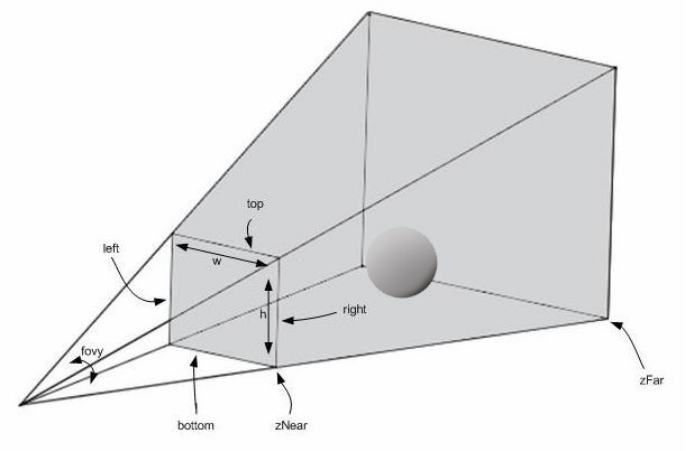


Figura 4-2 Projecção Perspectiva

4.2 Aparência

Para aumentar o realismo num jogo é introduzida iluminação no mundo virtual de modo a melhor simular um ambiente real com fontes de luz. São descritos abaixo os métodos de cálculo de iluminação mais comuns e a forma como estes estão integrados dentro da *Virtools*.

4.2.1 Modelo de reflexão de *Phong*

Descreve a interacção entre a luz e uma superfície, de acordo com as propriedades da luz incidente e do material constituinte da superfície. O cálculo de iluminação é efectuado para cada ***pixel*** da superfície.

Tanto a luz como o material da superfície são compostos por três componentes distintas, pelo que as componentes para a fonte de luz indicam a sua quantidade emitida e no material indicam a sua quantidade reflectida:

- **Componente Ambiente**: Componente que representa luz que já foi reflectida em todo o ambiente e está espalhada em todas as direcções. É representada por uma constante

$$I_{amb} = I_a K_a \tag{4.1}$$

I_a representa a intensidade da componente ambiente da luz e K_a o coeficiente de reflexão ambiente do material. A componente ambiente é reflectida pela superfície em todas as direcções.

- **Componente Difusa**: Fonte de luz emite luz com uma direcção e sentido específicos. A luz ao bater na superfície é reflectida igualmente em todas as direcções no caso de esta ser uma superfície difusa perfeita, ou seja, não é dependente da posição na qual está a ser visualizada. A intensidade da componente difusa da luz é dada por:

$$I_{diff} = I_i K_D \cos\theta \quad (4.2)$$

I_i representa a intensidade da fonte de luz, θ representa o ângulo formado entre o vector de luz incidente e a normal da superfície, K_D é uma constante que representa um coeficiente da quantidade de componente difusa que é reflectida pelo material.

Esta expressão pode ser reescrita pelo produto interno entre o vector Normal da superfície e o vector de luz incidente, estes que deverão estar normalizados.

$$I_{diff} = I_i K_D (L \bullet N) \quad (4.3)$$

L representa a direcção do vector vindo da fonte de luz e ***N*** a normal à superfície.

Caso hajam múltiplas fontes de luz, todas as suas contribuições deverão ser tomadas em conta.

$$I_{diff} = K_D \sum_n (I_n \bullet N) \quad (4.4)$$

- **Componente Especular:** Componente da luz com uma direcção definida e tende a reflectir numa determinada direcção **R** igual ao ângulo de incidência. A quantidade de componente especular visualizada depende da direcção de visualização.

  Pode-se interpretar a componente especular como o brilho numa determinada superfície.

A cor num certo ponto de uma superfície é determinada pela combinação destas três componentes através da seguinte equação:

$$I = I_a K_a + I_i [K_d (L \bullet N) + K_s \cos^n B] \quad (4.5)$$

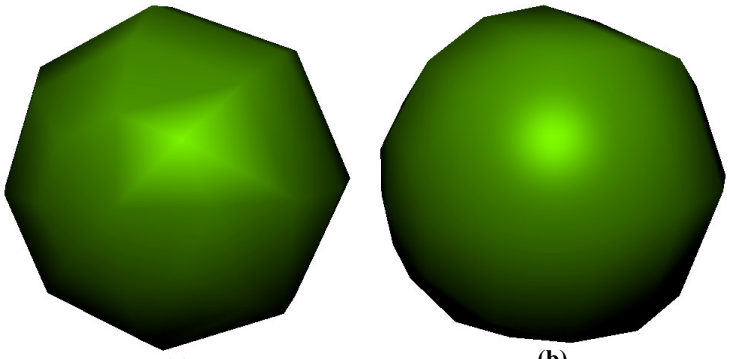
$$I = I_a K_a + I_i [K_d (L \bullet N) + K_s (R \bullet V)^n] \quad (4.6)$$

B é o ângulo entre o vector de visualização **V** e o vector de reflexão **R**, K_s é o coeficiente de reflexão especular e normalmente é uma constante que depende do material. O expoente ***n*** indica a quantidade de brilho de uma superfície

4.2.2 Modelo de iluminação de *Gouraud*

A iluminação é efectuada para cada vértice. São calculadas as normais para cada um dos vértices e de acordo com as normais, material e fontes de luz é calculada a cor do vértice utilizando o modelo de ***Phong***. As cores dos pixeis intermédios são depois calculadas através da interpolação da cor calculada nos vértices extremos. Consultar [19].

Para melhor se visualizar as diferenças entre os dois tipos de iluminação foi desenvolvida uma pequena demonstração de iluminação por **vértice** e por **pixel** recorrendo ao software ***RenderMonkey*** [20] e linguagem ***GLSL*** [21] cujo resultado é mostrado nas figuras abaixo.



(a)
Iluminação por vértice

(b)
Iluminação por pixel

Figura 4-31 Iluminação por pixel vs iluminação por vértice

A esfera apresentada contém apenas 112 triângulos pelo que é um modelo extremamente simplificado de modo a melhor realçar as diferenças entre os tipos de iluminação. Para a iluminação por vértice esta é renderizada a **1830** ***fps*** enquanto com iluminação por pixel é renderizada a **1609** ***fps***. Há uma perda total de mais de **200** ***fps***, no entanto pode-se verificar através das figuras uma melhoria substancial da qualidade da iluminação do modelo.

A *Virtools* não é em si um motor gráfico pelo que recorre às *API's* ***OpenGL*** ou ***Direct3D*** para fazer a renderização do conteúdo 2D/3D. Ambas as *API's* especificadas utilizam o modelo de iluminação de ***Gouraud*** dado que este é significativamente mais rápido pois o cálculo de cor é apenas feito para cada vértice. Sendo assim a *Virtools* apenas é dada a opção de renderizar sem modelo de sombreamento ou com modelo de *Gouraud*.

A definição das propriedades de uma determinada superfície em *Virtools* é feita através de **materiais**. A cada objecto está atribuído um **material**, que contém as três componentes, **ambiente, difusa** e **especular** referidas acima. Numa fonte de luz apenas se pode especificar a componente difusa da fonte de luz e se esta activa ou não a componente especular do material. A componente ambiente da luz não está associada a uma fonte de luz mas sim a todo o ambiente do mundo e pode ser especificada através do bloco de comportamento ***Set Ambient Light***.

4.3 Transformações geométricas

É através da utilização de transformações geométricas que todos os objectos existentes num jogo são movimentados, rodados ou modificados de tamanho. As transformações são efectuadas através de um sistema de coordenadas cartesiano, este que pode ser local ao objecto na qual se está a fazer a operação, ou global ao mundo virtual. A *Virtools* utiliza um sistema de coordenadas definido pela regra da mão esquerda, apesar dos sistemas de coordenadas diferirem de *OpenGL* para *Direct3D*, regra da mão direita versus regra da mão esquerda.

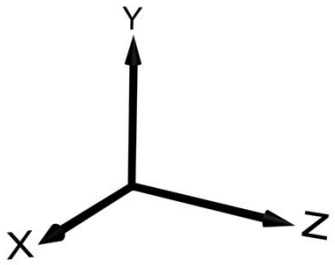


Figura 4-4 Sistema de coordenadas regra da mão esquerda

De seguida é feita uma breve descrição do tipo de transformações geométricas que podem ser aplicadas.

- **Translações:** Este tipo de transformação permite mover um objecto para uma nova posição através da especificação de um vector de deslocamento ou de uma nova posição. O vector deslocamento pode ser relativo ao sistema de coordenadas local ou global. Esta operação é efectuada através dos blocos de comportamento ***Translate*** ou ***Set Position***.

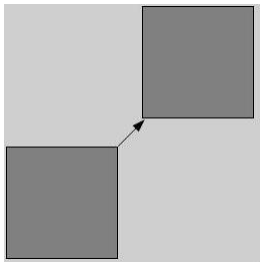


Figura 4-5 Translação

- **Rotações:** A orientação de um determinado objecto é alterada através de operações de rotação. A rotação, tal como a translação pode ser efectuada relativamente a um sistema de coordenadas local ou global. Existem diversas formas de aplicar uma rotação a um objecto pelo que é feita uma pequena descrição destas.

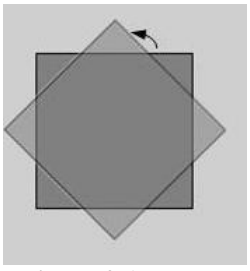


Figura 4-6 Rotação

  - **Ângulos de *Euler*:** Especifica uma nova orientação através da aplicação sucessiva de três rotações descritas pelos ângulos de *Euler*. Estes ângulos são os seguintes:
    - ***Roll*** especifica rotação no eixo global dos **zz's**
    - ***Pitch/Elevation*** especifica rotação no eixo global dos **xx's**
    - ***Yaw/Azimuth*** especifica rotação no eixo global dos **yy's**

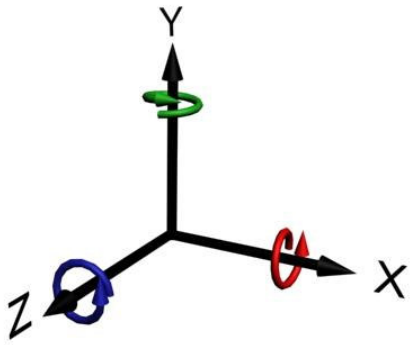


Figura 4-7 Ângulos de Euler

No entanto a utilização de ângulos de *Euler* faz surgir um problema denominado ***Gimbal Lock***. Este acontece quando há a perda de um grau de liberdade de rotação efectuando **sucessivas rotações de 90º** até que uma destas não ocorra devido ao alinhamento dos seus eixos. A razão para acontecer este problema é derivada do facto de os **ângulos de Euler avaliarem cada eixo independentemente numa ordem preestabelecida**. Tomando o seguinte exemplo, a ordem pela qual as rotações são avaliadas é xx, yy e zz e aplica-se uma rotação de 90º no eixo yy. Como a rotação ao eixo dos xx já foi avaliada, e não ocorreu nenhuma rotação este continua com a mesma orientação, assim quando é feita a rotação de 90º ao eixo dos yy, tanto o eixo dos yy como o eixo dos zz, vão sofrer alteração de orientação ficando o eixo dos zz agora alinhado com o eixo dos xx.

- **Quaterniões:** Através da utilização de quaterniões é resolvido o problema de ***Gimbal Lock*** isto porque as rotações com aqueles avaliam os três eixos em simultâneo para determinar a nova orientação. Um quaternião tem duas notações possíveis, notação de números complexos ou notação de vector 4D.
  - $w + xi + yj + zk$, **w**, **x**, **y** e **z** são escalares, **i**, **j**, **k** representam a componente imaginária
  - $[w, v]$, **v** é um vector **(x,y,z)** e **a** é um escalar

Os valores acima representados podem ser calculados da seguinte forma

$$w = \cos(\frac{\theta}{2}) \tag{4.7}$$

$$v = \sin(\frac{\theta}{2}) \times v' \tag{4.8}$$

Em que θ é o ângulo de rotação e **v'** é o eixo no qual é efectuada a rotação.

Apenas quaterniões unitários representam rotações, estes que estão contidos numa esfera de raio unitário contida num espaço 4D. Para uma descrição mais detalhada consultar [22].

A *Virtools* permite definir rotações tanto com ângulos de *Euler,* quaterniões, ou indicando directamente o ângulo e eixo no qual a rotação deve ser aplicada através dos blocos comportamento ***Set Euler Orientation***, ***Set Quaternion Orientation*** e ***Rotate*** respectivamente. Em todos os blocos comportamentais acima referidos é necessário igualmente optar pela transformação ser aplicada utilizando o sistema de coordenadas local ao objecto ou global ao mundo.

- E**scala:** Com uma transformação de escala é possível redimensionar um determinado objecto em relação a um ou mais dos seus eixos. Esta operação é efectuada na *Virtools* através do bloco de comportamento ***Scale***.

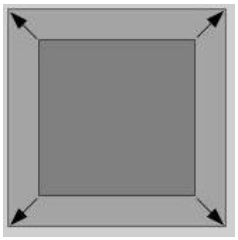


Figura 4-8 Escala

4.4 Detecção de Colisões

A *Virtools* fornece diversos mecanismos de detecção de colisões. É dada uma pequena descrição de quais os algoritmos existentes de detecção de colisões e como podem ser utilizados dentro da *Virtools*.

- **Colisão Esfera-Esfera:** Cada objecto está contido numa esfera envolvente, pelo que para efectuar o teste de colisão basta verificar se a distância entre objectos é ou não inferior à soma dos raios das suas esferas envolventes. Efectuado pelo bloco de comportamento ***Sphere Sphere Intersection***.

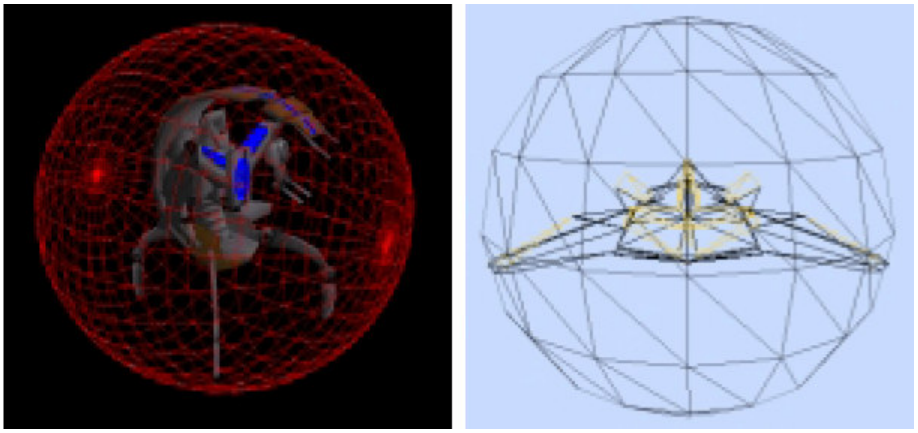


Figura 4-9 Volume Esfera

- **Colisão com *Axis Aligned Bounding Box*, *AABB*:** O objecto está contido num cubo envolvente alinhado com os eixos globais do mundo pelo que estes não podem ser rodados através das operações de rotação aplicadas aos objectos. Assim torna-se necessário recalcular as *AABB* para cada *frame*.

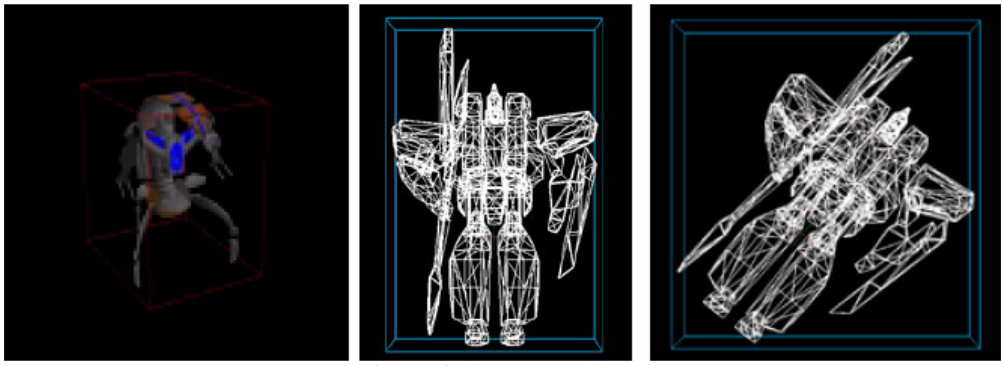


Figura 4-10 Volume *AABB*

- **Colisão com *Oriented Bounding Box*, *OBB*:** Técnica parecida à descrita acima, no entanto os cubos envolventes são orientados de forma a minimizar o espaço vazio, ou erro, provocado pela aproximação da forma do objecto ao cubo. Este processo é mais preciso que ***AABB***, no entanto é computacionalmente mais pesado.

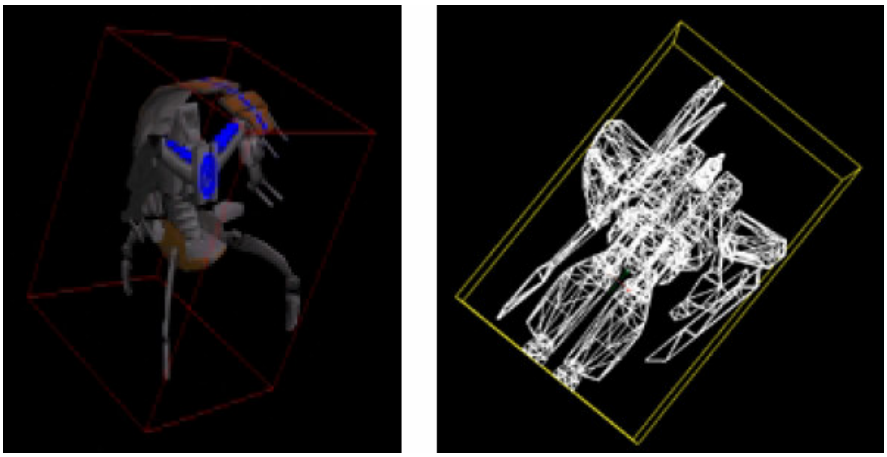


Figura 4-11 Volume *OBB*

- **Colisão Face-Face:** Teste de colisão entre os diversos triângulos constituintes de cada objecto. Técnica mais precisa, no entanto é computacionalmente pesada

Diversas técnicas são utilizadas em paralelo de modo a melhorar o cálculo de colisões, tal como a utilização de **múltiplos *OBB*** para cada parte do objecto em vez de apenas um, estes que poderão estar ligados hierarquicamente entre si.

Para o teste de colisão Face-Face, de modo a optimizar o número de cálculos, deverá ser realizada a detecção de colisões com uma malha alternativa ***low-poly***, com um número de triângulos significativamente menor, e não com a malha original de modo a que o número de triângulos testados ser inferior.

São igualmente utilizadas técnicas de particionamento espacial, tal como ***Binary Space Partitioning***, ***BSP***, de modo a que os testes de colisões não sejam feitos a todos os objectos de um cenário mas apenas àqueles que estão mais próximos do objecto a que se quer detectar colisão. Consultar [23].

A *Virtools* fornece diversos blocos de comportamento que permitem fazer colisões utilizando *Bounding Boxes* e/ou colisão de faces. Como exemplo são os blocos comportamentais ***Collision Detection***, ***Box Box Intersection***, ***Face Face Intersection*** e ***Box Face Intersection***.

4.5 Motor de Física

Define as bibliotecas responsáveis pela simulação das leis da física Newtoniana, calculando formas realistas de interacção entre os diversos objectos num mundo virtual A representação de um determinado objecto num motor físico costuma diferenciar da forma como este é representado visualmente no jogo. Tal como foi dito na secção sobre detecção de colisões, um objecto é constituído tanto por uma malha poligonal detalhada como por uma malha poligonal simplificada contendo apenas a informação geométrica essencial, esta última utilizada pelo motor de física para o cálculo de colisões e de física realista. A simulação de um objecto tipicamente é feita através de corpos rígidos (***rigid body***) com uma malha poligonal não deformável, no entanto torna-se cada vez mais comum a simulação de corpos com malhas deformáveis (***soft bodies***) tais como roupa ou balões. Para além de informação geométrica cada objecto passa a ter propriedades físicas tais como:

- Massa
- Posição do Centro de massa
- Atrito
- Elasticidade

Um motor de física define também restrições entre objectos, limitando desta forma o tipo de movimentos que estes possa efectuar. Na sequência de figuras abaixo são mostrados alguns exemplos de restrições normalmente disponibilizados pelos motores de física.

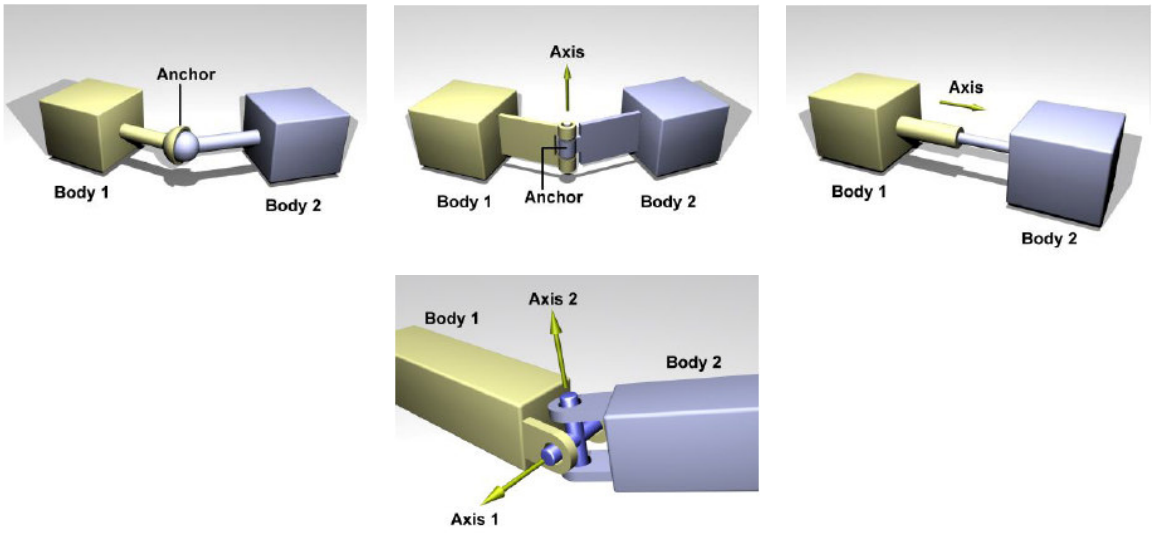


Figura 4-12 Restrições em motores de física

Para além de restrições simples, são utilizados sistemas de restrições tais como ***ragdolls***, bonecos de trapos. Estes são restrições associadas ao esqueleto de uma personagem, de modo a que esta, quando actuada por algum estímulo do jogo faça movimentos equivalentes aos efectuados por um

ser humano ou outro tipo de animal. Na ausência de estímulos, por exemplo morte da personagem, as restrições deverão continuar activas para que a queda deste seja a mais realística possível.

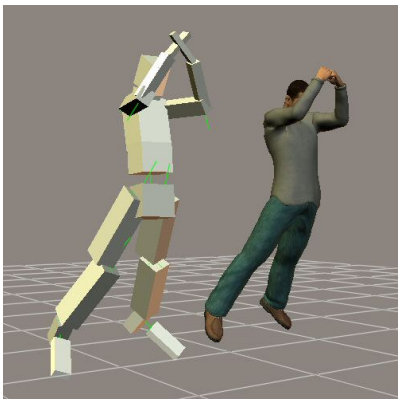


Figura 4-13 Sistema de *ragdolls*

Para efeitos de optimização, é usual num motor de física a desactivação de objectos caso estes estejam imóveis durante um determinado tempo e estes só são activos e tidos novamente em conta pelo motor de física caso haja uma colisão com estes. Desta forma os recursos do processador são melhor utilizados aumentando consequentemente a *frame rate* do jogo.

Exemplos de motores físicos são:

- ***Havok*** [24]
- ***ODE*** [25]
- ***Ageia PhysX*** [26]
- ***Newton Game Dynamics*** [27]
- ***Bullet Physics Library*** [28]

A plataforma *Virtools* contém um motor de física, ***Physics Pack***, baseado no motor ***Havok.*** Este permite a inclusão de comportamentos físicos nos objectos de um jogo criado em *Virtools*, atribuir-lhes propriedades físicas e restrições simples, *ragdolls* e sistema físico de um carro e definir constantes do mundo virtual tal como a gravidade. Isto pode ser efectuado através da utilização dos blocos de comportamento disponibilizados pelo motor de física. Exemplos de blocos de comportamentos de *Physics Pack* são:

- ***Physicallize***
- ***Set Physics Ball Joint***
- ***Set Physics Joint***
- ***Set Physics Constraint***
- ***Modify Physics Globals***
- ***Physics Car***

4.6 Efeitos especiais

Nesta secção são descritos os modos de funcionamento de efeitos especiais que estão integrados nas aplicações em *Virtools*. No entanto será antes feita uma pequena descrição do que é um

FrameBuffer. Este é o ultimo estágio numa *pipeline* de renderização, onde são guardados os resultados de uma série de operações efectuadas em fragmentos tais como:

- **Aplicação de textura**
- **Testes de recorte, *alpha*, *stencil* e remoção de superfícies ocultas**
- **Cálculo de nevoeiro**

Por definição um fragmento corresponde à conversão de uma geometria em elementos quadrados e tem correspondência directa com os ***pixeis*** constituintes do ***FrameBuffer***. Um fragmento é composto por informação de:

- **Cor**
- **Profundidade**
- **Valor de coordenadas de textura**

Um ***Framebuffer*** é constituído pelos seguintes buffers:

- ***Color Buffer*****:** Guarda a cor constituinte de cada ***pixel***. Esta cor costuma ser expressa em ***RGBA, Red, Green, Blue, Alpha***
- ***Depth Buffer*****:** Guarda um valor de profundidade para cada ***pixel***
- ***Stencil Buffer*****:** Normalmente utilizado como máscara, para restringir o desenho a partes específicas do ecrã
- ***Accumulation Buffer*****:** Contém dados ***RGBA*** tal como o ***Color Buffer***. É tipicamente utilizado para acumular uma série de imagens numa imagem final composta por todas as anteriores

Tendo esta informação em conta procede-se de seguida à explicação dos efeitos especiais integrados nas aplicações em *Virtools*.

4.6.1 Sombras *Stencil*:

Será feita uma breve descrição do processo de criação de sombras volumétricas utilizando o ***Stencil Buffer***. Primeiramente é abordada a forma tradicional de criação de ***sombras stencil*** [29] e de seguida é descrito o método **"Carmack's Reverse"** que corrige certos aspectos da primeira abordagem.

Definem-se como **oclusores** todos os objectos que provoquem sombra, sendo os restantes objector definidos como **receptores de sombra**. Na figura abaixo encontra-se um cenário, visto de cima, em que a esfera é o elemento oclusor e o polígono é o receptor de sombra. De forma a simplificar a situação não será tida em conta a sombra gerada pelo polígono.

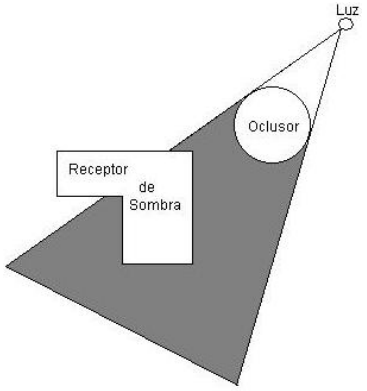


Figura 4-14 Volume de Sombra

A região a cinzento representa o volume de sombra. Esta é obtida extrudindo as arestas de silhueta da esfera a partir do ponto de vista da fonte de luz até uma distância finita ou infinita.

O cenário é depois renderizado e, para cada fragmento é feito o seguinte teste que irá modificar os valores do *stencil buffer*:

- Renderizar face frontal do volume de sombra. Se teste de profundidade passar, então incrementa-se valor correspondente de *stencil*, caso contrario não se faz nada. Desactivar escrita tanto para o *color buffer* como para o *depth buffer*.
- Renderizar face traseira do volume de sombra. Se teste de profundidade passar, então decrementa-se valor correspondente de stencil, caso contrario não se faz nada. Desactivar escrita tanto para o *color buffer* como para o *depth buffer*.

Abaixo é mostrada uma figura que exemplifica os passos referidos acima.

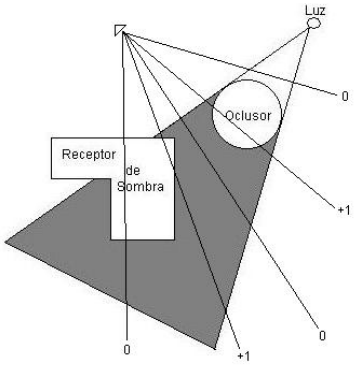


Figura 4-15 Modificação Stencil Buffer 1º algoritmo, 1 objecto

Dado que é utilizado a informação de *z-pass/depth pass*, este algoritmo é conhecido por técnica de *Depth Pass Stencil Shadow Volume*. O algoritmo é também aplicável para múltiplos oclusores como se pode verificar na figura abaixo.

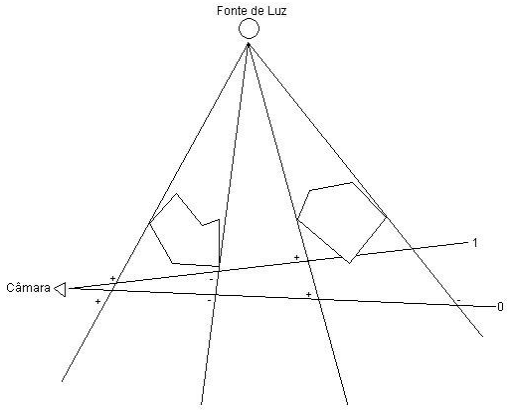


Figura 4-16 Modificação Stencil Buffer 1º algoritmo, múltiplos objecto

No entanto há uma falha neste algoritmo, caso a câmara esteja dentro do volume de sombra provoca valores anómalos no *stencil buffer* fazendo com que as sombras não sejam construídas correctamente.

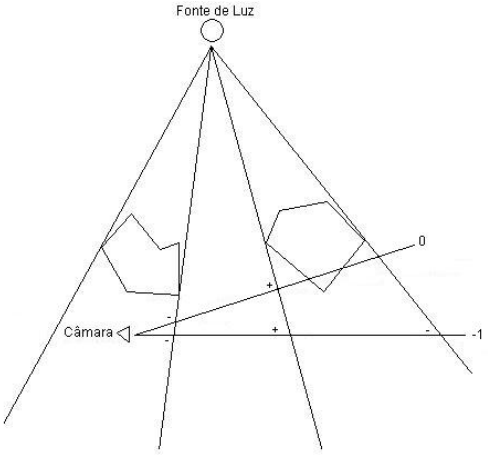


Figura 4-17 Erro no preenchimento de Stencil Buffer

De modo a contornar esse erro, ***Jonh Carmack*** [30] modificou o algoritmo de modo se utilizar não o teste de ***depth pass*** mas sim o teste de ***depth fail*** para modificar o conteúdo do *stencil buffer*. Os novos passos do algoritmo, conhecido como ***"Carmacks Reverse"*** são:

- Renderizar face traseira. Se teste de profundidade falha então incrementar o valor correspondente do *stencil buffer*, caso contrario não se faz nada. Desactivar escrita tanto para o *color buffer* como para o *depth buffer*.
- Renderizar face frontal. Se teste de profundidade falha então decrementar o valor correspondente do *stencil buffer*, caso contrario não se faz nada. Desactivar escrita tanto para o *color buffer* como para o *depth buffer*.

Deste modo as sombras são geradas correctamente, independentemente da posição da câmara como se pode verificar na figura abaixo

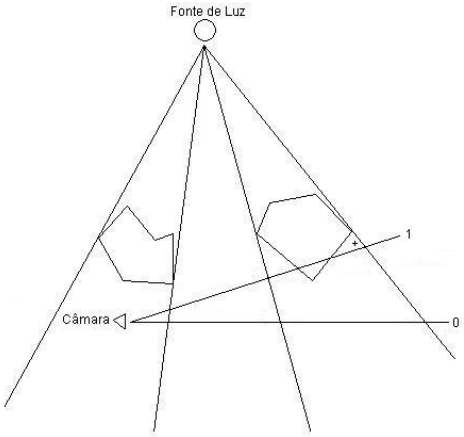


Figura 4-18 Preenchimento de Stencil buffer com algoritmo de Carmack

Este algoritmo de geração de sombras é utilizado dentro de uma aplicação em *Virtools* através do bloco de comportamento ***ShadowStencil***, em que neste é indicado:

- Fonte de luz a ser utilizada
- Comprimento do volume de sombras
- Câmara pode ou não penetrar no volume de sombra

Para além da utilização do referido bloco de comportamento é também necessário indicar explicitamente quais os objectos oclusores existentes no cenário.

4.6.2 Reflexões planares

Considera-se uma superfície plana reflectora ideal em que o ângulo de incidência da luz é idêntico ao ângulo de reflexão fazendo com que a imagem de um objecto reflectida é o próprio objecto reflectido segundo o plano reflector. Caso o plano reflector se encontre em $Y = 0$ basta aplicar uma transformação de escala $S(1,-1,1)$ a todos os objectos, ou seja, mantêm-se as coordenadas **(xx,zz)** e espelha-se as coordenadas **yy**.

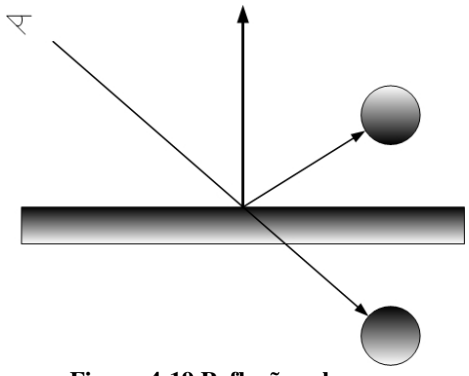


Figura 4-19 Reflexões planares

Para um plano reflector qualquer translada-se e orienta-se este de modo a passar na origem e coincidir com $Y = 0$, aplicar uma transformação de escala $S(1,-1,1)$ e inverter as duas primeiras operações.

$$R = [R()T()]^{-1} S(1,-1,1)[R()T()] \quad (4.9)$$

É necessário restringir os objectos reflectidos à superfície reflectora. Para isso é utilizado o *stencil buffer* como máscara, em que é colocado um valor superior a zero nas posições do *stencil buffer* correspondente ao plano reflector e o restante conteúdo é deixado a zero, para que quando os objectos reflectidos são desenhados fiquem restritos ao plano reflector.

No caso de existirem objectos localizados atrás do plano reflector é necessário definir um plano de recorte coincidente com aquele de modo a quando se reflecte a cena, os objectos em que estejam atrás do plano reflector não sejam desenhados em frente a este.

O algoritmo de reflexões planares pode ser utilizado dentro da *Virtools* através do bloco de comportamento ***Planar Reflection***, no qual é necessário especificar o objecto planar que deverá obter características reflectoras, configurar correctamente os seus materiais, e indicar quais os objectos que deverão ser reflectidos.

4.6.3 Sistemas de Partículas

Um sistema de partículas é basicamente um conjunto de pontos 3D no espaço. A representação visual destes pontos pode ser um simples vértice no espaço, um *billboard* (quadrado constantemente alinhado com a orientação da câmara) ou um objecto 3D.

Cada partícula é definida por um conjunto de propriedades, estas que podem variar com o tipo de sistema de partículas:

- Posição
- Tamanho
- Velocidade
- Peso
- Tempo de vida
- Cor (incluindo canal alfa)

As partículas são geradas através de um emissor. Este controla factores como o número de partículas a ser geradas, velocidade inicial de cada partícula e define os parâmetros iniciais destas. O número de partículas a serem geradas e atribuição de velocidade inicial é um processo estocástico, ou seja, é atribuída uma componente aleatória a estes de modo a que o efeito final não seja demasiado rígido mas flua de uma forma mais caótica. Por exemplo:

$$NumParticulasGeradas = NumParticulasCte + (\mathrm{var}\,iânciaEmissor * numAleatório) \quad (4.10)$$

O número de partículas geradas a cada instante é a soma de uma componente constante com uma componente que irá indicar a variância, ou seja o número de partículas a mais ou a menos que podem ser geradas. Para isso *numAleatório* é um número aleatório entre [-1,1]. Para uma descrição detalhada sobre criação de sistemas de partículas consultar [31].

A plataforma *Virtools* permite a criação de diversos tipos de sistemas de partículas podendo variar a forma do emissor, por exemplo forma esférica ou forma discoide. Cada partícula pode ser representada, como dito anteriormente, através de pontos, *billboards* ou objectos 3D.

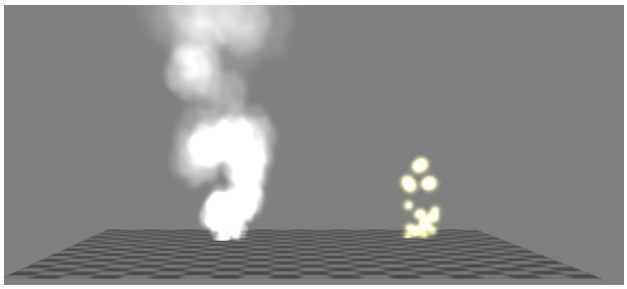


Figura 4-20 Sistema de partículas

Os blocos comportamentais para os diversos sistemas de partículas são:

- **CubicParticleSystem**
- **CurveParticleSystem**
- **CylindricalParticleSystem**
- **DiscParticleSystem**
- **LinearParticleSystem**
- **ObjectParticleSystem**
- **PlanarParticleSystem**
- **PointParticleSystem**
- **SphericalParticleSystem**

5 Desenvolvimento de Aplicações de Realidade Virtual

Uma vez feita a integração de todo os dispositivos, procede-se agora à descrição do desenvolvimento dos jogos. É desenvolvido um jogo de Ténis de Mesa/*Ping-Pong* de nome ***PongVR*** que faz uso do *tracker* magnético para o controlo de uma raquete, é configurada uma câmara com estereoscopia activa pelo que a sua posição é controlada através de gestos com uma luva calçada na mão esquerda e orientação controlada pelo giroscópio interno dos óculos. No entanto esta aplicação não faz uso de todo o potencial das luvas, pelo que se decide criar mais um jogo em que é calçado um par de luvas *5DT*, cada uma com um receptor de *tracker* magnético. Com a informação obtida de posição e orientação de cada mão, aliada com a informação de flexão dos dedos pelas luvas movimenta-se e manipula-se um par de mãos virtuais que têm como finalidade a interacção com um piano virtual, sendo o nome do jogo ***PianoVR***. O presente capítulo tem assim o intuito de explicar o desenvolvimento de cada uma das aplicações.

5.1 *PongVR*

O objectivo desta aplicação é o de simular um jogo de ténis de mesa. Para isso o jogador utiliza o receptor do *tracker* magnético para movimentar e orientar a raquete no mundo virtual.

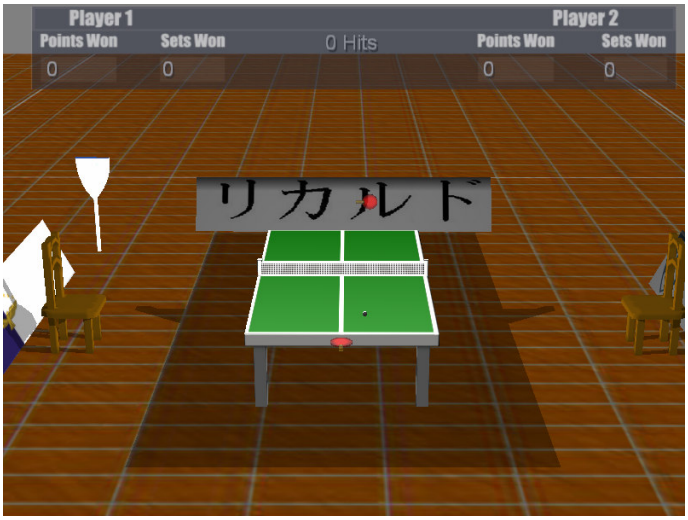


Figura 5-1 ***PongVR***

A visualização do jogo é feita através da utilização dos óculos *eMagin Z800* em que a câmara virtual é orientada através da utilização do giroscópio dos óculos e movimentada através de um código gestual interpretado de uma luva *5DT* calçada na mão esquerda. De notar que a simulação física do jogo é efectuada através do *plugin* de física disponibilizado pela *Virtools.*

A arquitectura simplificada do jogo é a seguinte:

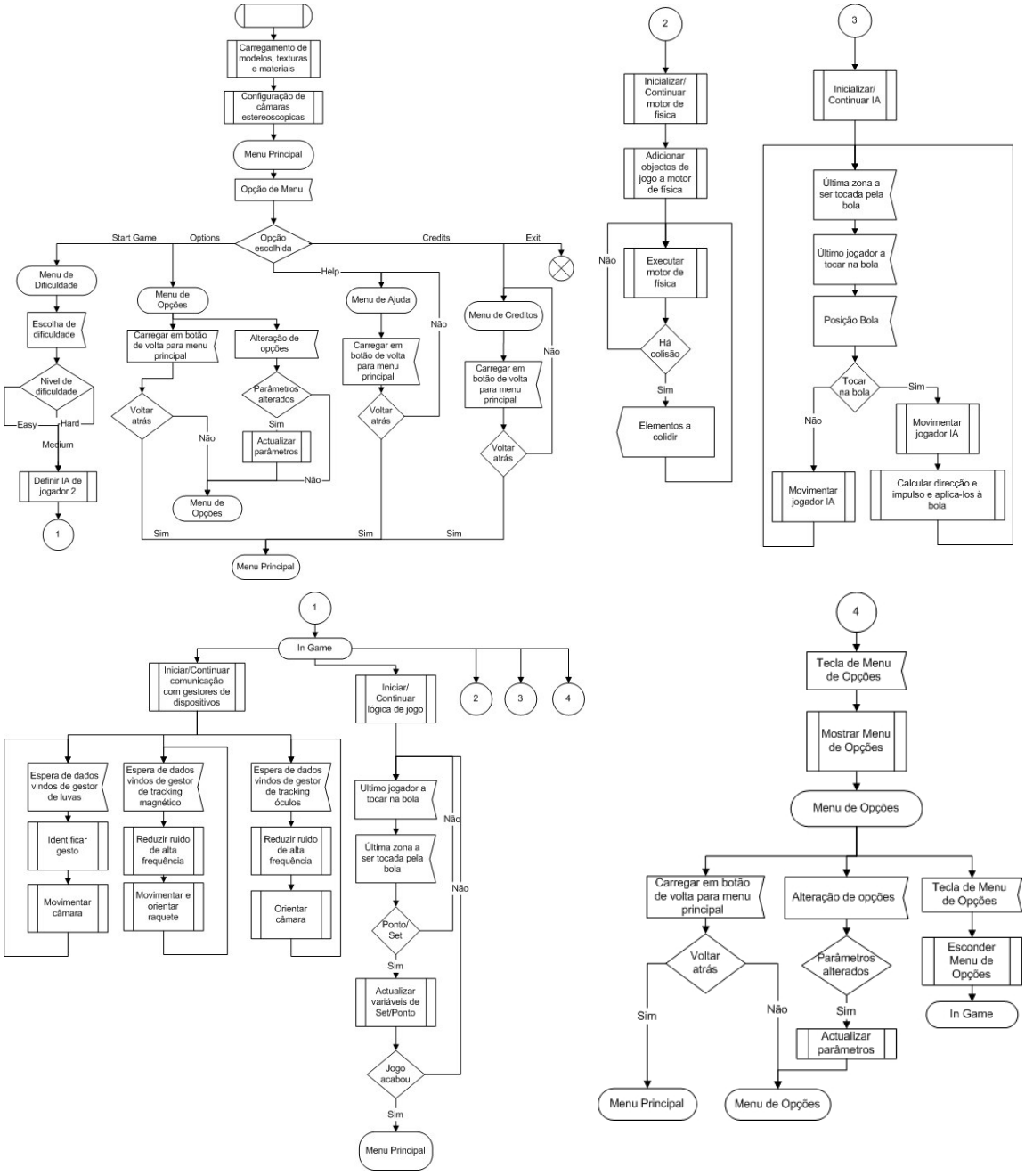


Figura 5-2 Arquitectura simplificada de *PongVR*

Neste capítulo é apenas descrita a arquitectura de jogo a partir do estado ***In Game***, sendo que o resto é devidamente explicado em anexo. A título exemplificativo são colocados em anexo alguns dos *scripts* constituintes de ***PongVR***.

Ao iniciar o estado ***In Game*** são lançados *scripts* que são responsáveis por:

- Aquisição e tratamento de dados de gestores de dispositivos
- Motor físico de jogo
- Gestão de lógica de jogo
- Cálculo de Inteligência Artificial de adversário

5.1.1 Aquisição e tratamento de dados de gestores de dispositivos

É criado um *script* que está em constante comunicação tanto com o gestor das luvas, gestor do *tracker* e gestor dos óculos através da utilização dos comportamentos ***GloveBlock***, ***TrackerBB*** e ***eMaginBB***. Os dados retirados são depois passados por um filtro passa-baixo simples de modo a eliminar ruído de altas-frequência. A sequência de operações é a seguinte:

Obtenção/Tratamento de dados de *tracker*

1. Retira posição e orientação de *tracker* magnético
2. Determinação de tempo de execução de bloco de comportamento
3. Passar dados por filtro passa baixo
    a. Inserir dados num vector de tamanho predefinido
        i. Se vector cheio, eliminar elemento mais antigo antes da inserção
    b. Calcular média aritmética de valores contidos no vector
4. Determinar nova orientação
    a. Aplicar nova orientação a eixo de coordenadas local
5. Determinar nova posição
    a. Aplicar nova posição a eixo de coordenadas do mundo

Obtenção/Tratamento de dados de óculos

1. Retirar orientação de *tracker* óculos
2. Determinação de execução de bloco de comportamento
3. Passar dados por filtro passa baixo
    a. Inserir dados num vector de tamanho predefinido
        i. Se vector cheio, eliminar elemento mais antigo antes da inserção
    b. Calcular média aritmética de valores contidos no vector
4. Determinar nova orientação
    a. Aplicar nova orientação a eixo de coordenadas local

Obtenção/Tratamento de dados de luvas

1. Retira gesto de luvas
2. Determinação de tempo de execução de bloco de comportamento
3. Translação de câmara de acordo com gesto

Figura 5-3 Pseudo-código de scripts de aquisição de dados de PongVR

Dado que a orientação e posição do receptor do *tracker* é relativa ao seu emissor, não são utilizados os dados directamente como vêm do filtro passa-baixo mas sim é calculada a variação entre os dados actuais e os calculados na *frame* anterior para que deste modo se possa colocar e orientar a raquete num estado inicial e só depois esta ser afectada pelas variações na posição e

orientação do receptor do *tracker*. Em anexo encontra-se a sequência de comportamentos para o cálculo da nova orientação e posição da raquete.

Foi pensada e testada a hipótese de utilização de quaterniões para a definição da orientação, dado que estes são imunes a problemas de *Gimbal Lock,* no entanto os ângulos fornecidos já são uma orientação final e são calculados internamente pelos dispositivos em forma de quaterniões logo como não é necessário fazer composição de orientações a utilização de quaterniões não se justifica.

É importante notar que a aplicação dos ângulos de *Euler* calculados não é feita directamente ao objecto 3D que representa a raquete mas sim a uma entidade 3D denominada *3D Frame*, que não tem uma presença visual, apenas contem informação de transformação no mundo virtual.

Para a configuração estereoscópica da câmara de jogo é efectuado o procedimento já explicado em 3.3. A movimentação desta é feita através da utilização de gestos com uma luva calçada na mão esquerda ou através de um suporte adicional criado para teclado. A interpretação dos gestos é feita recorrendo ao parâmetro de saída *gesture* do comportamento *GloveBlock* em que são mapeados 6 dos 16 gestos disponibilizados. O procedimento tomado para a orientação de câmara é em todo semelhante ao *tracker* magnético mas considerando apenas os ângulos ***Yaw/Azimuth*** e ***Pitch/Elevation*** dado que o movimento de ***Roll*** não é normalmente utilizado na orientação de uma câmara de jogo.

5.1.2 Motor físico de jogo

É utilizado o motor ***Physics Pack*** da *Virtools* para a simulação de física de jogo em ***PongVR***.

Um objecto 3D para ser tido em conta pelo motor físico necessita de ser integrado no motor de física através do bloco de comportamento ***Physicallise***. Através deste comportamento são indicados parâmetros como massa do objecto e malha poligonal a considerar na detecção de colisões.

Após este processo deixa de ser possível efectuar operações de translação e rotação directamente ao objecto 3D, pelo que terão de ser aplicadas a uma entidade 3D que não esteja ligada ao motor físico e depois é ordenado ao objecto físico que siga esta entidade, através do bloco de comportamento ***Motion Controller***.

Após todos os objectos estarem integrados no motor físico este fica automaticamente responsável pelo comportamento e interacção entre todos os objectos “fisicalizados”.

Os *scripts* de inserção e transformação de objectos no motor físico encontram-se em anexo.

5.1.3 Gestão de lógica de jogo

É agora mostrada e explicada a máquina de estados para a atribuição de pontuações aos jogadores e de seguida são detalhados os três níveis de inteligência artificial que podem ser atribuídos ao oponente.

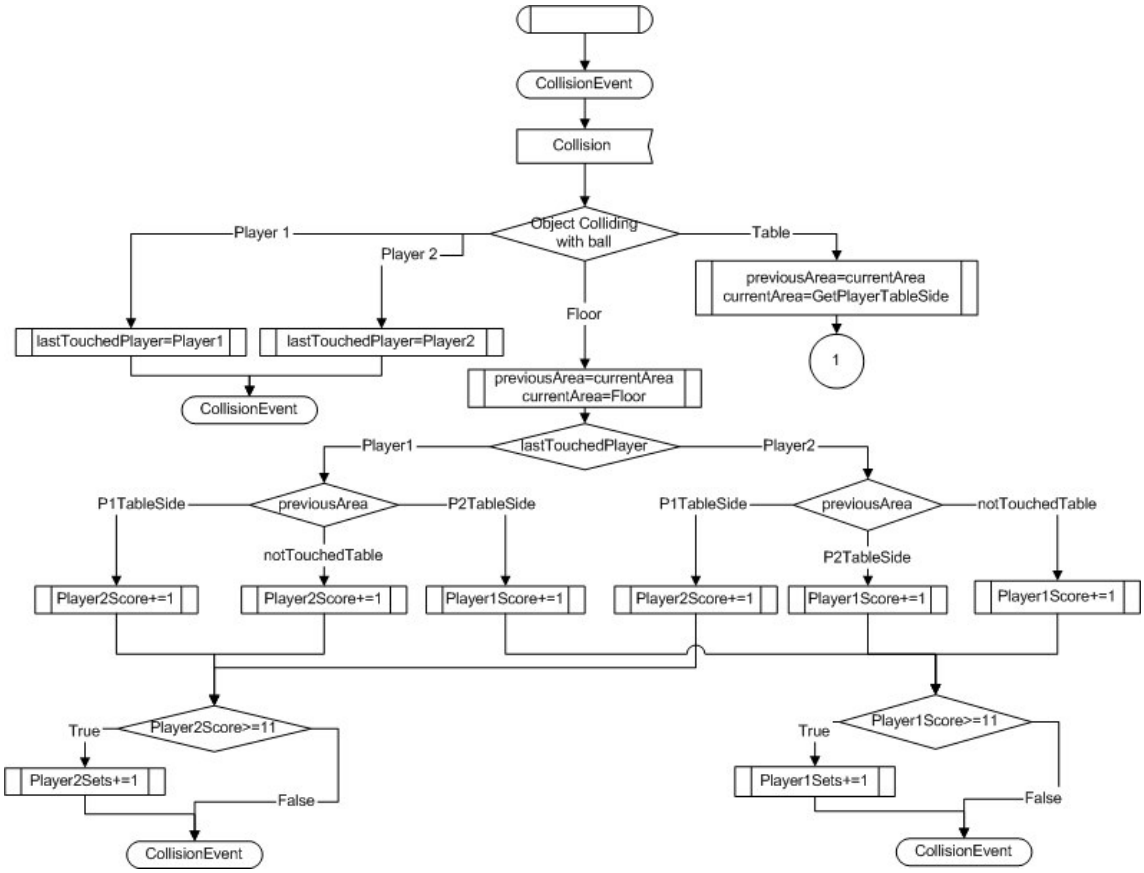


Figura 5-4 Máquina de estados de atribuição de pontuação em *PongVR* (1)

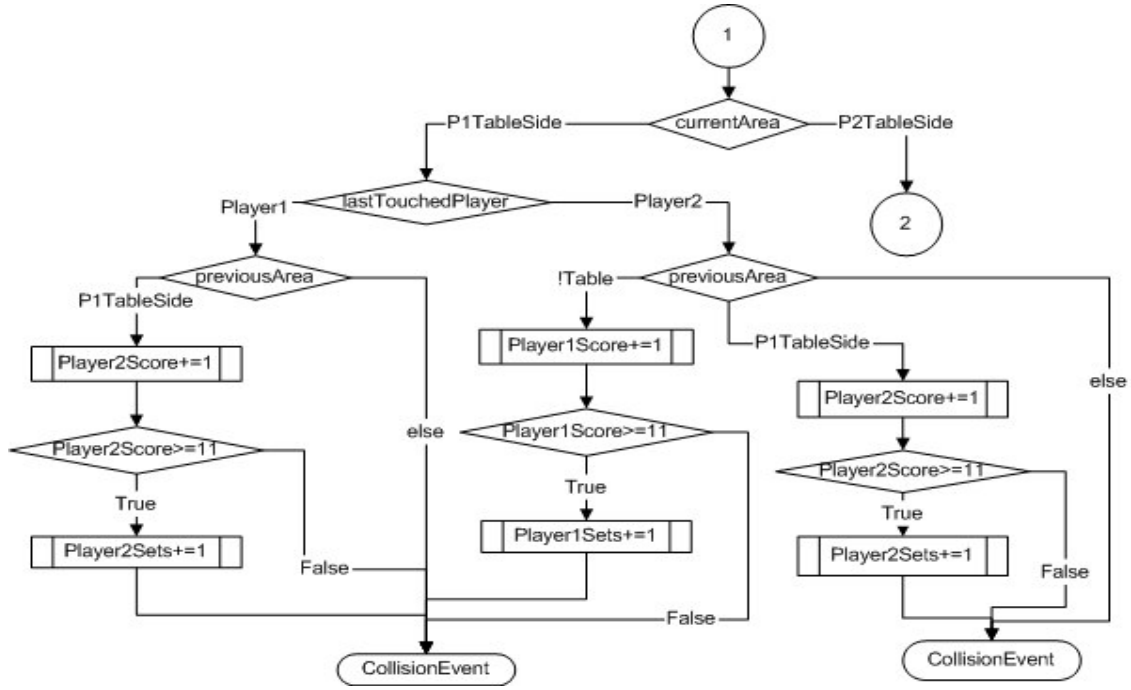


Figura 5-5 Máquina de estados de atribuição de pontuação em *PongVR* (2)

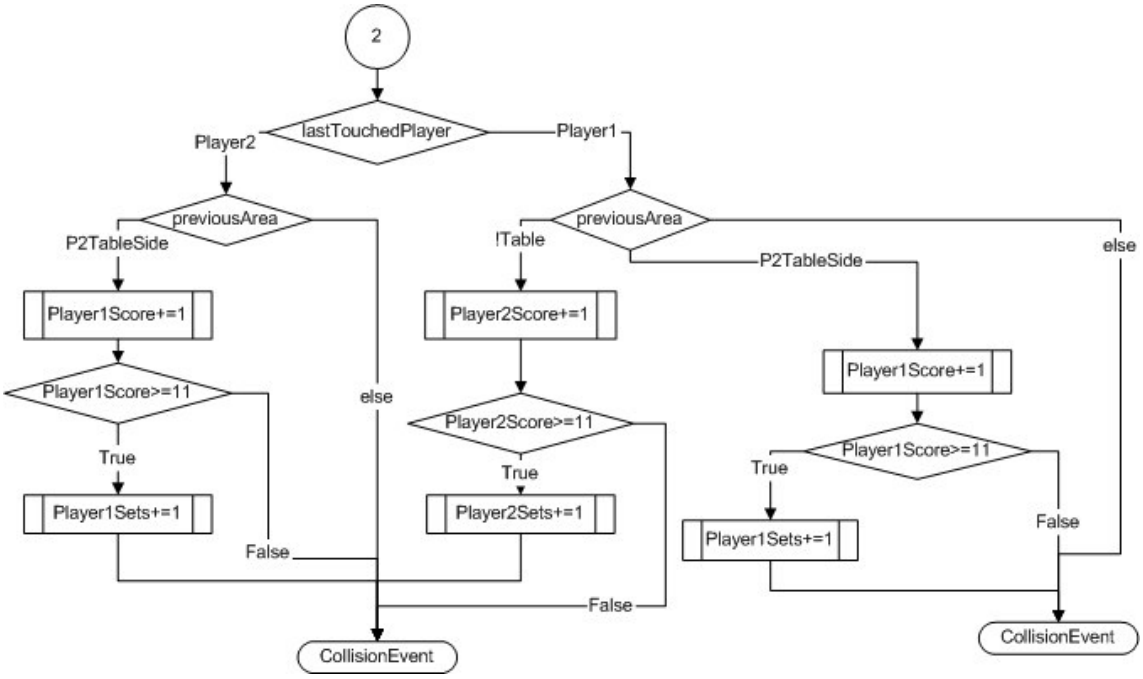


Figura 5-6 Máquina de estados de atribuição de pontuação em *PongVR* (3)

A máquina de estados, tal como mostram as figuras, é despoletada sempre que haja uma colisão da bola com um de quatro elementos cénicos, jogador 1, jogador 2, mesa ou chão. Caso seja jogador 1 ou 2, há apenas uma atribuição do identificador do jogador em questão a uma variável que indica o ultimo jogador que tocou na bola. De acordo com as regras oficiais um jogador não pode tocar mais que uma vez seguida na bola, no entanto esta regra é ignorada por uma questão de oferta de maior jogabilidade ao jogador 1.

Figura 5-7 Áreas de jogo

Caso a colisão seja entre a bola e a mesa ou chão são verificados factores como quem foi o último jogador a tocar na bola, quais as duas últimas duas áreas da mesa que foram tocadas pela bola e, de acordo com essa informação, há ou não a atribuição de ponto.

Foi acrescentado um contador de toques, em que o gestor de lógica mostra ao jogador mensagens de incentivo e altera o aspecto visual da bola, associando a esta um rasto e um sistema de partículas, à medida que este aumenta o número de toques consecutivos na bola sem perder para o adversário.

As mensagens apresentadas são as seguintes:

- ***Cool Combo*** para 3 toques consecutivos
- ***Awesome Combo*** para 7 toques consecutivos
- ***Ping Pong God Combo*** para 9 toques consecutivos

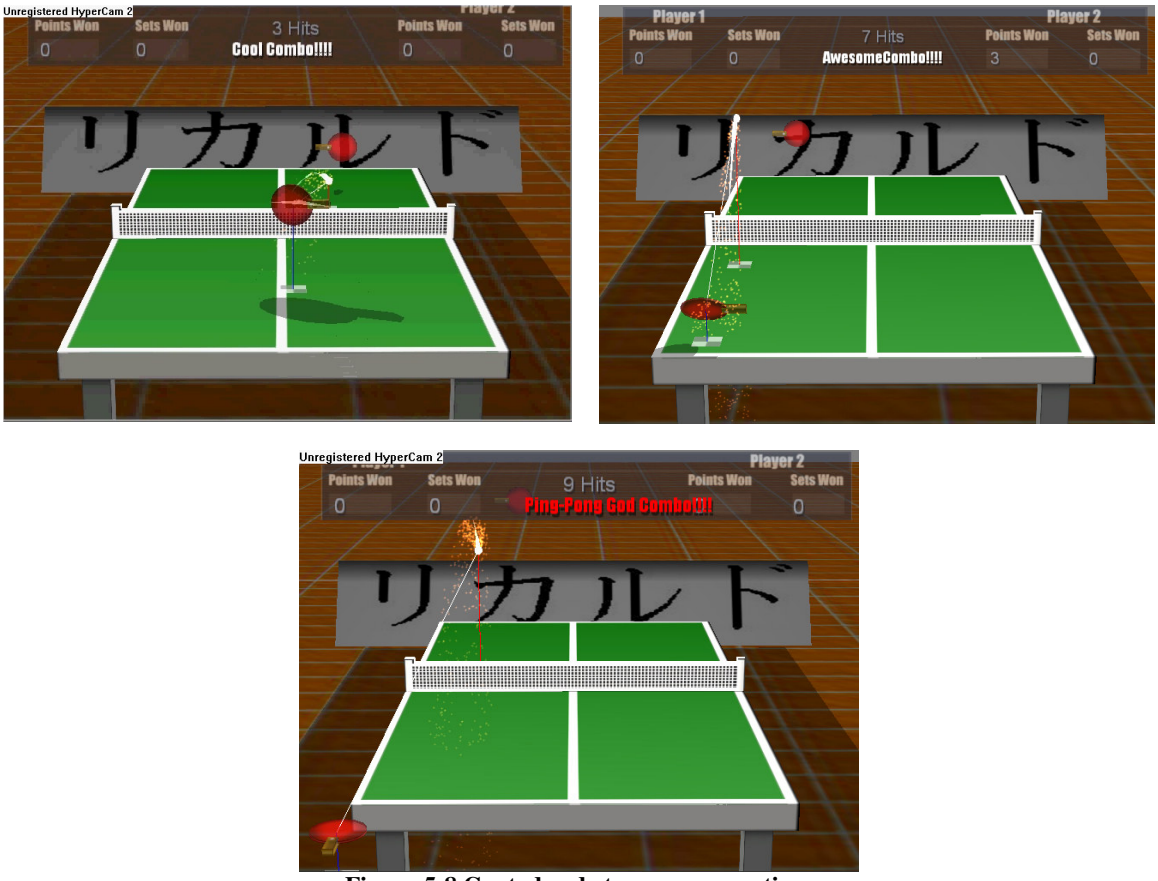


Figura 5-8 Contador de toques consecutivos

5.1.4 Cálculo de Inteligência Artificial de adversário

Para o jogo são elaborados três níveis de dificuldade para o adversário, fácil, médio e difícil.

No grau fácil, o adversário envia a bola de volta na direcção do jogador com uma elevada componente vertical de modo a que bola faça uma trajectória maior para que assim seja dado mais tempo de resposta ao jogador facilitando a sua jogada.

Em relação ao grau médio, o oponente irá ripostar sempre para uma marca situada no centro do campo do jogador, sem a componente vertical elevada colocada no grau de dificuldade anterior.

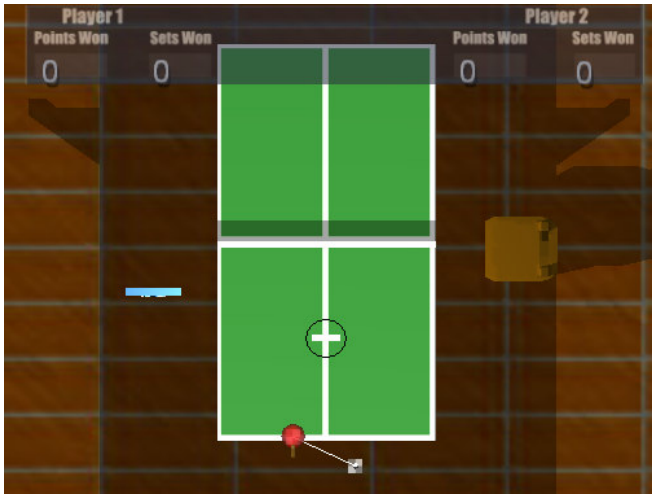


Figura 5-9 Localização marca – Nível de dificuldade médio

No grau de dificuldade difícil, o adversário irá enviar a bola para o canto oposto do campo ao qual o jogador esteja, *p.e.* caso o jogador esteja com a raquete junto às redes do seu lado esquerdo, o adversário irá atirar a bola para o canto direito mais afastado da rede.

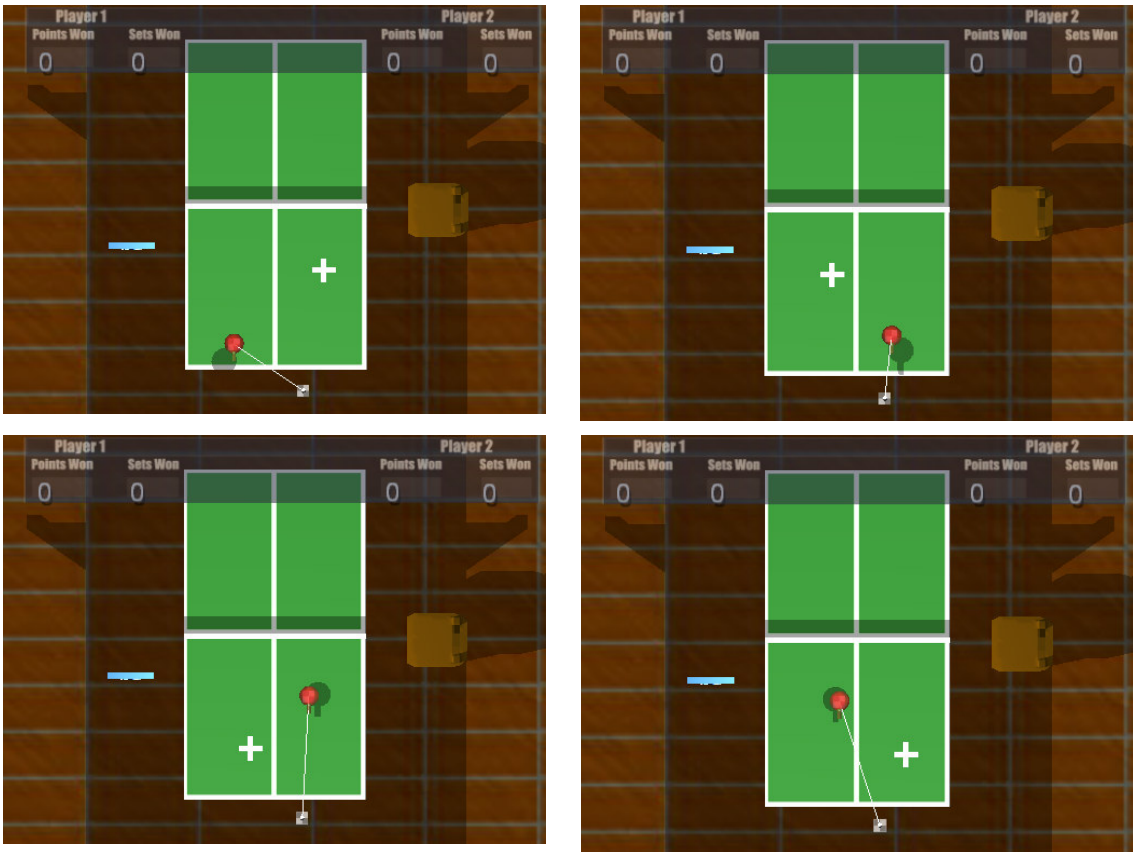


Figura 5-10 Localização Marcas - Nível de dificuldade difícil

Para todos os graus de dificuldade o impulso aplicado pelo adversário à bola e movimento do adversário são efectuados de forma idêntica. Enquanto a bola se encontra no campo do jogador

humano o adversário apenas efectua movimentos num plano xy atrás da mesa de modo a se alinhar com os movimentos da bola efectuados dentro do espaço da mesa. Só quando a bola toca na área adversária de forma válida é que este efectua um deslocamento até à posição da bola, aplica-lhe um impulso e retorna à sua posição inicial. O impulso aplicado é proporcional tanto à distância entre a bola e a marca colocada na área de jogo como à velocidade da bola no momento do impacto por parte do jogador 1.

O impulso é calculado através da variação de momento linear.

$$I = \Delta p \tag{5.1}$$

$$I = m\Delta v \tag{5.2}$$

$$I = m \times (v_{final} - v_{inicial}) \tag{5.3}$$

Em que I é o impulso a aplicar à bola, Δp define a variação do momento linear, m indica a massa da bola e Δv indica a variação de velocidade desta.

A componente $v_{inicial}$ é dada pela velocidade actual da bola. É necessário descobrir qual a velocidade final que a bola terá obter de modo a conseguir viajar para o campo contrário e embater no ponto especificado pela marca colocada naquele.

Para isso são utilizadas as equações que definem o movimento parabólico de uma partícula,

$$x = \| v_0 \| \times \cos\theta \times t \tag{5.4}$$

$$y = \| v_0 \| \times \sin\theta \times t - \frac{1}{2} \times \| g \| \times t^2 \tag{5.5}$$

Em que x é o alcance desejado, $\| v_0 \|$ é a norma da velocidade a aplicar à bola, θ é o ângulo formado entre a bola e a mesa e $\| g \|$ é q norma da aceleração gravítica.

A equação de trajectória é obtida igualando a componente y a zero.

$$0 = \| v_0 \| \times \sin\theta \times t - \frac{1}{2} \times \| g \| \times t^2 \tag{5.6}$$

$$0 = t \times (\| v_0 \| \times \sin\theta - \frac{1}{2} \times \| g \| \times t) \tag{5.7}$$

$$t = 0 \vee t = \frac{2 \times \sin\theta \times \| v_0 \|}{\| g \|} \tag{5.8}$$

$$x = \| v_0 \| \times \cos\theta \times \frac{2 \times \sin\theta \times \| v_0 \|}{\| g \|} \tag{5.9}$$

$$x = \frac{\| v_0 \|^2 \times \sin 2\theta}{\| g \|} \tag{5.10}$$

$$\| v_0 \| = \sqrt{\frac{x \times \| g \|}{\sin 2\theta}} \tag{5.11}$$

Com o valor de $\| v_0 \|$ calculado é possível determinar um valor aproximado do impulso necessário a aplicar à bola fazendo

$$v_{final} = \| v_0 \| \tag{5.12}$$

Após a estrutura base do jogo estar completa, é notado que mesmo com a utilização dos óculos VR, não se consegue ter uma noção perfeita de profundidade do mundo virtual, pelo que se torna difícil acertar na bola na primeira jogada, e agravada na situação de ripostar a uma jogada feita pelo adversário. De modo a tentar minimizar esta dificuldade são criados mecanismos adicionais, para além de um sistema de sombras, que visam ajudar o jogador a ter uma melhor noção da relação espacial bola/raquete.

O primeiro mecanismo incluído é o de desenhar uma linha que liga, durante todo o jogo, a raquete do jogador à bola. Esta linha tem o objectivo de fornecer informação visual extra, juntamente com as sombras projectadas, ao jogador da distância entre a bola e a raquete.

O segundo mecanismo é o de desenhar mais duas linhas, a primeira entre uma marca móvel e a raquete; a marca tem as mesmas coordenadas (x,z) da raquete mas com coordenada vertical (y) fixa à altura da mesa. Em relação à segunda linha, esta é desenhada de forma análoga mas agora entre a bola e a sua respectiva marca. O objectivo deste conjunto de duas marcas é o de fornecer ajuda visual adicional sobre a relação de alturas entre a bola e a raquete.

Um terceiro mecanismo é o de fazer um ajuste automático da posição da raquete caso esteja numa distância próxima da bola. Este mecanismo, apesar de continuar disponível, não está activo dado que piorava a jogabilidade na medida em que confunde o jogador com os auto-ajustes efectuados.

O quarto e último mecanismo a ser implementado é o de, caso a bola se encontre num volume preestabelecido envolvente à raquete do jogador, esta é atraída para a raquete.

Infelizmente, com os mecanismos descritos acima, continua a ser difícil ripostar às jogadas do adversário.

O aspecto visual do jogo a decorrer com ajudas activas é mostrado na figura abaixo.

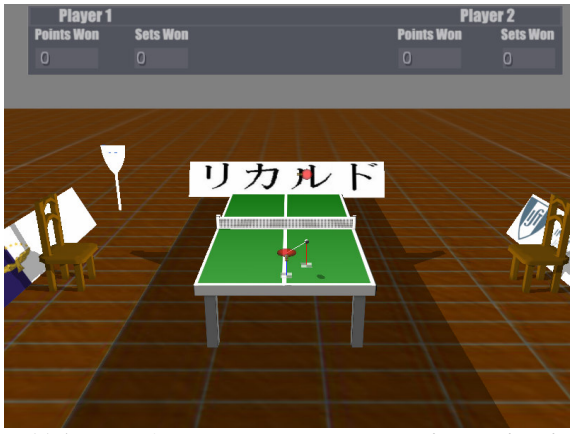


Figura 5-11 Aspecto *in-game* de *PongVR* com ajudas visuais activas

Encontra-se em anexo um manual de instruções que cobre os aspectos do jogo que não foram referenciados neste capítulo, tal como interpretação do *HUD, Heads-Up Display*, e campos do menu de opções.

5.2 *PianoVR*

O objectivo desta aplicação é o de explorar melhor as funcionalidades do par de luvas *5DT* dado que, na aplicação anterior o seu uso foi extremamente reduzido. Para tal é decidido, a criação de um piano virtual, que possa ser manipulado por um par de mãos virtuais orientadas e movimentadas individualmente através de um receptor de *tracking* para cada. Mais uma vez, a visualização é feita através da utilização dos óculos *VR eMagin Z800 3D Visor* com uma configuração idêntica à apresentada no jogo *PongVR*.

Figura 5-12 *PianoVR*

Abaixo é ilustrada a arquitectura simplificada do jogo.

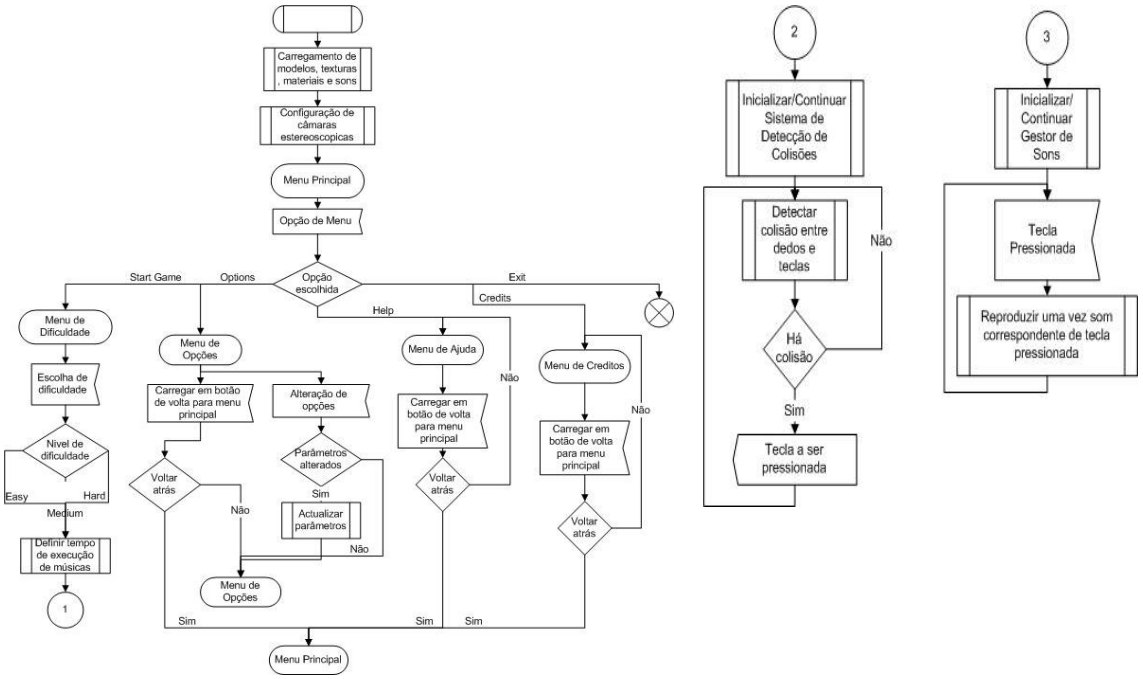


Figura 5-13 Arquitectura simplificada de *PianoVR* (1)

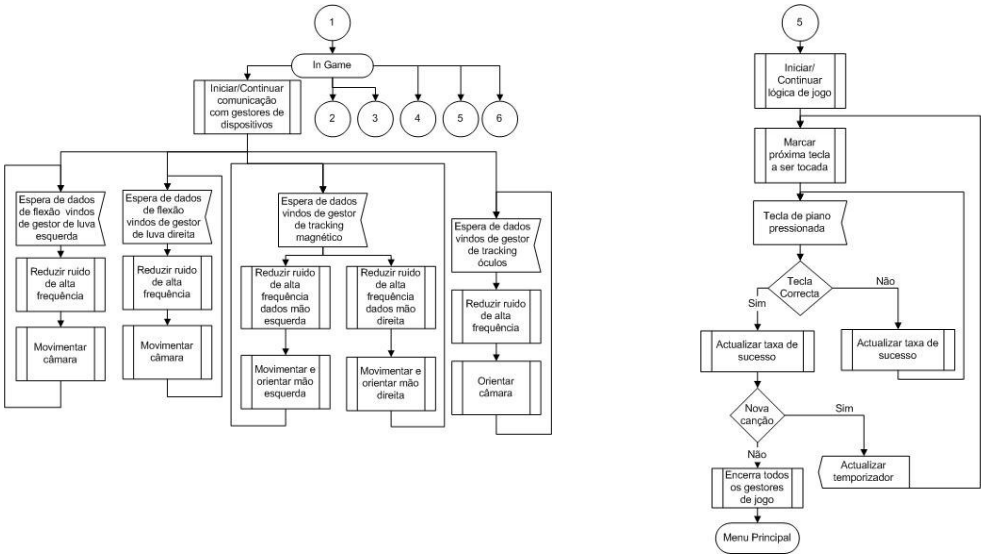


Figura 5-14 Arquitectura simplificada de *PianoVR* (2)

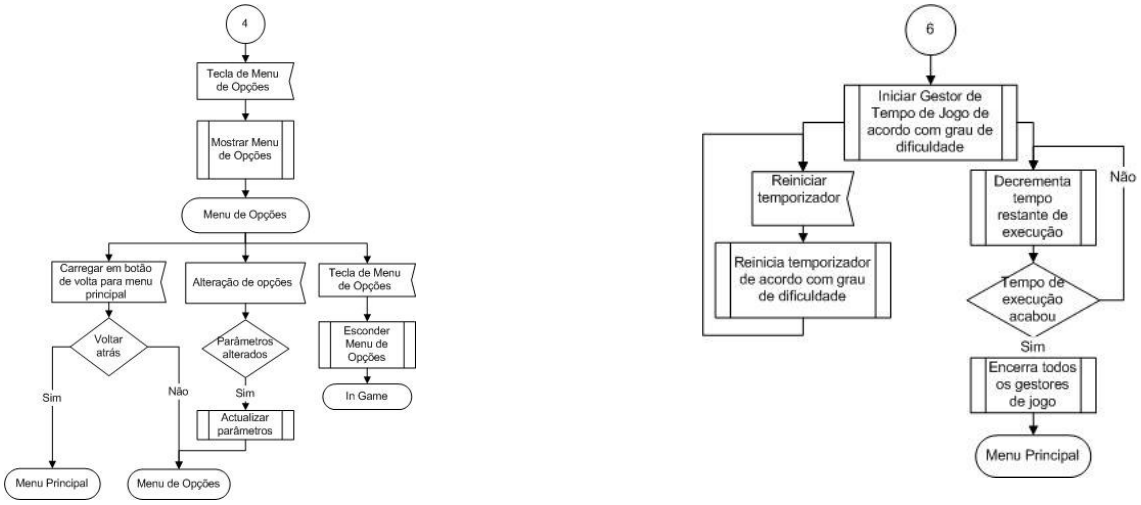


Figura 5-15 Arquitectura simplificada de *PianoVR* (3)

Tal como em *PongVR*, a análise da arquitectura será feita apenas ao estado ***In Game*** pelo que uma descrição mais detalhada dos restantes componentes é efectuada em anexo.

Ao passar para o estado ***In Game*** são iniciados *scripts* que são responsáveis por:

- Aquisição e tratamento de dados de gestores de dispositivos
- Gestão de detecção de colisões
- Gestão de lógica de jogo
- Gestão de reprodução de sons
- Gestão de temporizador

5.2.1 Aquisição e tratamento de dados de gestores de dispositivos

O processo de utilização dos *trackers* é idêntico ao mostrado na aplicação *PongVR*, com a pequena diferença de se estar a utilizar dois blocos de comportamento *TrackerBB*. Isto deve-se ao facto de ser necessário retirar informação de dois receptores, um localizado na mão direita e o outro localizado na

mão esquerda. Ambos os valores retirados de cada um dos *trackers* são passados, de forma idêntica, por um filtro passa-baixo, de modo a atenuar o ruído de alta-frequência.

Das luvas é retirada informação de flexão para cada sensor. Esta informação é tratada e depois é aplicado o novo movimento calculado a um sistema de ossos previamente implementado nas mãos virtuais. Assim é criada uma malha poligonal na forma de uma mão e reveste-se esta com ossos, processo denominado por *skinning.* Esta tarefa é feita através do software *3D Studio Max* e exportada para *Virtools* para o seu formato nativo. A criação de ossos é essencial, dado que é através destes que a malha poligonal se deforma de modo a simular a movimentação de uma mão real. Os ossos são transformados dentro da aplicação através dos dados retirados dos sensores de ambas as luvas. Para mais informação visualizar [32].

Abaixo é mostrada uma figura com o mapeamento e identificação de ossos. Note-se que é apresentada uma figura da mão esquerda, pelo que a mão direita é um espelho daquela.

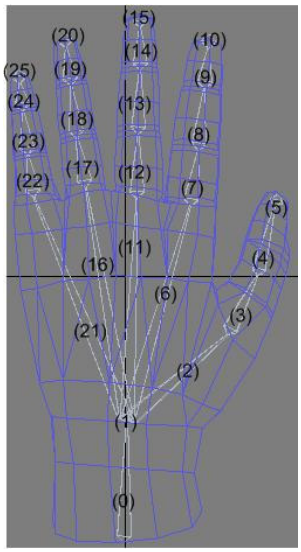


Figura 5-16 Mapeamento de ossos na mão virtual

Através da tabela 4-1 é mostrado o mapeamento entre os ossos da mão esquerda e os sensores da luva *5DT*. Não esquecer que esta contém apenas cinco sensores.

Tabela 5-1 Mapeamento ossos-sensores para luva mão esquerda

Mão esquerda	**3**	**7**	**12**	**17**	**22**
Sensor A	X				
Sensor B		X			
Sensor C			X		
Sensor D				X	
Sensor E					X

De seguida é mostrado mapeamento entre os sensores da luva da mão direita e ossos da mão virtual. Note-se que foram desprezados os valores obtidos pelos sensores localizados nas zonas interdigitais.

Tabela 5-2 Mapeamento ossos-sensores para luva mão direita

Mão direita	3	4	7	8	12	13	17	18	22	23
Sensor 0	X									
Sensor 1		X								
Sensor 3			X							
Sensor 4				X						
Sensor 6					X					
Sensor 7						X				
Sensor 9							X			
Sensor 10								X		
Sensor 12									X	
Sensor 13										X

O modo como são tratados os dados retirados dos sensores para ambas as mãos é de certa forma análogo ao como é feito para os dados do *tracker*. Os dados de cada um dos sensores são passados por um filtro passa-baixo, para minimizar ruído em alta-frequência, depois o valor filtrado é comparado com o recebido na *frame* anterior e é calculado o novo movimento a aplicar em cada um dos ossos.

1. Retirar flexão de sensores da luva
2. Determinação de tempo de execução de bloco de comportamento
3. Passar dados por filtro passa baixo
    b. Inserir dados num vector de tamanho predefinido
        i. Se vector cheio, eliminar elemento mais antigo antes da inserção
    c. Calcular média aritmética de valores contidos no vector
4. Para cada sensor, calcular variação de flexão
5. Aplicar variação de flexão a eixo de coordenadas local de osso correspondente

Figura 5-17 Pseudo-código de aquisição de informação de flexão das luvas

Em anexo é mostrada a sequência de *scripts* que efectua estas operações para a luva da mão esquerda. Para a mão direita o procedimento é idêntico, apenas com o acréscimo do número de sensores a tratar. O *script* de movimentação da mão direita contém no entanto uma pequena diferença: ao início é feita uma calibração da distância inicial entre as mãos virtuais, de modo a que haja correspondência directa entre a distância entre os dois receptores e a distância entre as mãos virtuais.

De seguida é explicado como é feita a interacção entre as mãos virtuais e o piano virtual.

5.2.2 Gestão de detecção de colisões

Neste jogo, ao contrário do anterior, não é utilizado o motor físico disponibilizado pela *Virtools*, dado que este é pouco flexível. Todo o mecanismo de interacção das mãos virtuais com o piano virtual foi desenhado de raiz. A cada tecla está associado um volume de colisão. Através da intersecção dos dedos com este volume de colisão é gerado um deslocamento fixo, independentemente da

velocidade de penetração no volume, colocando a tecla numa **posição de tecla premida**. A tecla só volta à sua **posição de repouso** quando deixar de haver intersecção entre esta e os dedos.
Quando uma tecla é premida é lançada uma mensagem internamente indicando apenas informação de qual a tecla em que ocorreu colisão. É irrelevante enviar informação de qual foi o dedo que tocou na tecla.

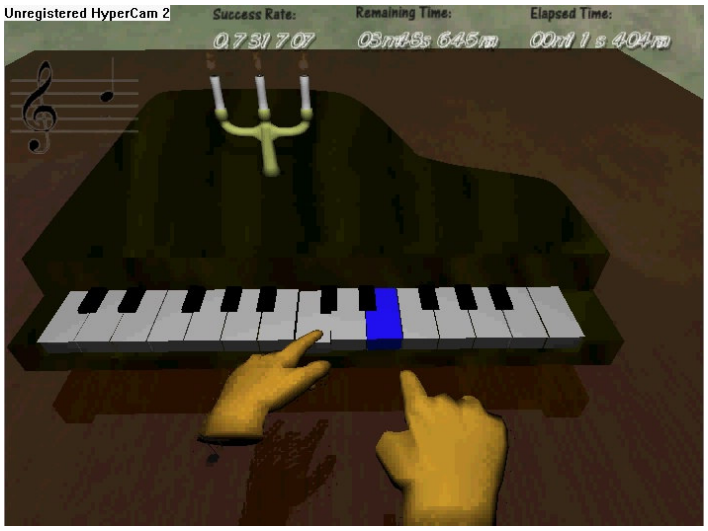


Figura 5-18 Colisão dedo-tecla

5.2.3 Gestão de reprodução de sons

Ao se iniciar o jogo, são carregados em memória os diversos sons de duas oitavas de piano.

Quando é recebida uma mensagem interna de colisão é reproduzido apenas uma vez o som correspondente à tecla pressiona, dado que se pretende simular um piano e não um órgão.

5.2.4 Gestão de lógica de jogo

Com o objectivo de melhorar o nível de interactividade do jogo, é pedido ao utilizador que toque uma sequência de teclas de uma música previamente registada num período de tempo determinado pelo nível de dificuldade escolhido. Para isso há um gestor de lógica a monitorizar todas as acções efectuadas entre as mãos e o piano. Caso o jogador consiga tocar todas as músicas dentro do tempo o gestor de lógica indica que o jogo é ganho, caso contrário o utilizador perde e terá de recomeçar.

É também calculada pelo gestor uma **taxa de sucesso**, isto é, a relação entre o número de teclas premidas na ordem correcta e o número total de teclas pressionadas, esta que vai sendo actualizada em tempo real e mostrada ao jogador no ***HUD*** do jogo.

5.2.5 Gestão de temporizador

Como foi indicado anteriormente, o utilizador dispõe de um tempo finito para tocar uma sequência de teclas. Este tempo, determinado pelo grau de dificuldade, tem um gestor próprio para que seja independente do processamento da lógica de jogo. A função do gestor é de actualizar o tempo passado desde o início de execução de determinada música, e enviar uma mensagem interna caso seja ultrapassado o tempo estipulado. Esta mensagem é recebida pelo gestor de lógica do jogo para

que este termine o jogo. O gestor de lógica comunica igualmente com o gestor de temporizador de modo a que quando uma música seja completamente tocada este reinicie a sua contagem.

Tal como para o jogo *PongVR*, está colocado em anexo para o *PianoVR* um manual de instruções que cobre os aspectos do jogo que não foram referenciados neste capítulo, tal como interpretação do *HUD*, e campos do menu de opções

6 Medição de Desempenho

Para determinar o desempenho de utilização dos dispositivos, inseridos no contexto de uma aplicação em *Virtools*, são desenvolvidos dois novos blocos de comportamento.

- ***Get_System_Time*:** Bloco de comportamento que fornece informação do tempo de sistema em *UTC*, *Universal Time Co-ordinated*, todavia a resolução temporal fornecida por este é baixa apresentando tempos em milissegundos.
- ***Get_Precise_Time*:** Bloco de comportamento desenvolvido em conjunto com um Gestor, que fornece informação temporal ao programador com resolução temporal ao nível de µ segundos.

As medições de desempenho são efectuadas utilizando ***Get_Precise_Time*** dado que este apresenta melhor resolução temporal. O tempo fornecido por ***Get_Precise_Time*** é o tempo desde a activação do Gestor quando é iniciada a aplicação *Virtools* até à activação corrente do bloco de comportamento. Este tempo é internamente calculado pelo Gestor recorrendo directamente ao registo de contador do relógio do processador [33].

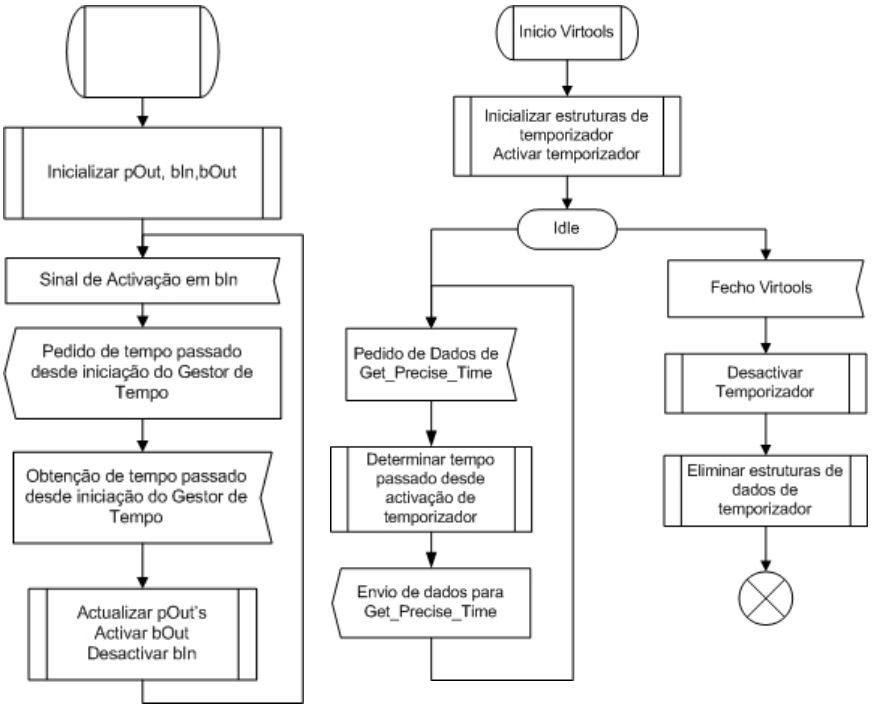

Inicializar pOut, bIn,bOut
Sinal de Activação em bIn
Pedido de tempo passado desde iniciação do Gestor de Tempo
Obtenção de tempo passado desde iniciação do Gestor de Tempo
Actualizar pOut's
Activar bOut
Desactivar bIn
Inicio Virtools
Inicializar estruturas de temporizador
Activar temporizador
Idle
Pedido de Dados de Get_Precise_Time
Determinar tempo passado desde activação de temporizador
Envio de dados para Get_Precise_Time
Fecho Virtools
Desactivar Temporizador
Eliminar estruturas de dados de temporizador

Figura 6-1 Funcionamento de *Get_Precise_Time* (à esquerda) + Gestor (à direita)

Tendo uma medição precisa do tempo disponível dentro da *Virtools* é possível determinar o tempo de captura dos dados dos diversos dispositivos e com isso fazer uma avaliação quantitativa de desempenho destes.

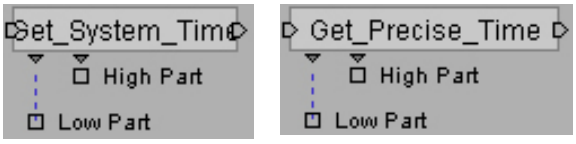


Figura 6-2 Aspecto visual de Get System Time e Get_Precise_Time

Ambos os blocos comportamentais contêm dois ***pOuts*****,** Para ***Get_System_Time*** indicam:

- **Inteiro** de nome **LowPart** indica 32 bits mais à esquerda do tempo em ***ms*** passado em *UTC*
- **Inteiro** de nome **HighPart** indica 32 bits mais à direita do tempo em ***ms*** passado em *UTC*

No caso de ***Get_Precise_Time*** os ***pOuts*** indicam:

- **Inteiro** de nome **LowPart** indica 32 bits mais à esquerda do tempo em ***µs*** passado desde a activação do Gestor de Temporizador
- **Inteiro** de nome **HighPart** indica 32 bits mais à esquerda do tempo em ***µs*** passado desde a activação do Gestor de Temporizador

O bloco de comportamento ***Get_Precise_Time*** é integrado nas aplicações *PongVR* e *PianoVR* e é feita uma medida temporal imediatamente antes e depois da utilização dos blocos de comportamento *TrackerBB*, *GloveBlock* e *eMaginBB* que efectuam a comunicação com o Gestor dos dispositivos de *tracking* magnético, luvas e *tracking* de óculos, e através da subtracção deste termos é calculado um tempo de execução do bloco de comportamento. Para além desta informação em *Virtools*, ambos os gestores de comunicação com as luvas, *tracker* magnético e *tracker* de óculos, são modificados de modo a que estes internamente façam também uma medida do tempo de cálculo de dados directamente nos dispositivos através da medição temporal ao início e final da função que faz a recolha dos dados. Esta funcionalidade não estava disponível na primeira versão dos gestores.

Com os novos dados temporais fornecidos pode-se saber o tempo total desde que a informação é pedida ao dispositivo e fica disponível ao programador em *Virtools*.

Abaixo é mostrada uma tabela com valores médios em diversas situações. Para o cálculo dos valores médios é utilizada uma população de 100 amostras obtidas, executando as aplicações num computador portátil com um processador *Intel T2400 Core Duo* a 1.8 *GHz*, 1024 *MB* de *RAM* e uma placa gráfica *ATI X1600* com 128 *MB* de *VRAM* dedicadas.

Tabela 6-1 Desempenho de dispositivos imersivos

	Empty Scene	***PianoVR***	***PongVR***
GloveBlock (µs)	8	345	56
TrackerBB (µs)	8	970	45
eMaginBB (µs)	41	42	200
Glove Manager (µs)	123	123	123
Tracker Manager One Receiver (µs)	22000	-	22000
Tracker Manager Two Receivers (µs)	33000	33000	-
eMagin Manager (µs)	54000	54000	54000

De modo a melhor se fazer a avaliação dos valores dados acima é necessário fornecer informação sobre o grau de complexidade de cada uma das aplicações e redefinir o conceito de tempo de latência. O tempo de latência como foi indicado anteriormente indica o tempo necessário a efectuar o transporte de dados entre o emissor e o receptor.

A aplicação *PongVR* contém um total de 752 blocos de comportamento em 47 *scripts*, contém um total de 49 objectos 3D, utiliza cerca 4.35MB em memória de texturas e é executada a aproximadamente 30 *fps*. Para a aplicação *PianoVR*, esta contém 9831 blocos de comportamento em 90 *scripts*, contém um total de 103 objectos 3D, utiliza cerca de 5.69MB em memória de texturas e é executada a aproximadamente 30 *fps*.

Assim para uma análise de pior caso é analisado o caso da aplicação *PianoVR* dado que utiliza um maior número de dispositivos e contém um muito maior número de blocos de comportamento. No entanto pode-se observar que o tempo de obtenção dos dados pelos gestores é independente da complexidade do jogo apenas variando do tempo de execução dos blocos de comportamento.

6.1 *Polhemus Fastrak*

A utilização de *tracking* magnético em *Virtools* produz um valor total de ~**34ms** que é a soma entre a componente que dada internamente pelo gestor e pela execução do bloco de comportamento:

O resultado interno do gestor fica um pouco acima do valor apresentado na descrição oficial do *tracker* dado que para uma frequência de amostragem de **60Hz** e tempo de latência de **4ms** soma um total de **20.67 ms**. A discrepância de **13.33 ms** entre o valor real e o valor apresentado deve-se ao tempo de cálculo interno dos dados por parte do módulo de interface/ filtro que não é contabilizado na informação oficial e também à introdução de tempo de latência inerente à execução das aplicações dentro do ambiente *Virtools*.

Pode-se definir um patamar no qual está localizada a tecnologia utilizada. O tempo que está a ser utilizado como referência é obtido no caso de utilização de dois receptores pelo que os valores expressos na **Tabela 1-1** são referentes a tempos óptimos logo com apenas um par emissor receptor.

Como se pode verificar o valor obtido de ~34ms está num patamar intermédio, em comparação com as outras tecnologias apresentadas.

6.2 Luvas *5DT Ultra*

De seguida é analisado o valor apresentado para as luvas *5DT*. Dado que são ligações *USB* separadas, não há um aumento do tempo total de utilização passando de uma para duas luvas. Este facto é visível pela semelhança dos valores indicados pelo Gestor das Luvas, tanto para a aplicação *PongVR*, que apenas contém a ligação a uma luva, como para a aplicação *PianoVR* que utiliza duas luvas. Ambos os valores apresentados pelo Gestor rondam os **123 µs.**

A fonte oficial da *5DT* apenas fornece a frequência de amostragem das luvas, indicando um mínimo de **75Hz** para o caso de se estar a utilizar o modelo que contém *tracking* incorporado, pelo que o valor real de frequência de amostragem das luvas que são utilizadas na tese em principio é maior em algumas ordens de grandeza, assim não é possível efectuar o cálculo de uma quantidade temporal teórica de modo a fazer uma comparação com o valor real.

De acordo com a **Tabela 1-2** e comparando com os valores apresentados na **Tabela 2-1**, pode-se apenas verificar que as luvas *5DT* se encontram num patamar intermédio em termos de frequência de amostragem, e com uma resolução idêntica aos exemplos apresentados.

6.3 Óculos *eMagin Z800 3D Visor*

Para o seguimento proveniente do giroscópio dos óculos, segundo [12] este possui uma velocidade de refrescamento de **125Hz**, com um tempo de latência do dispositivo de 4ms, Os dados são fornecidos com uma resolução de 1grau. O tempo teórico total de utilização é assim **12 ms**.

Há uma discrepância entre o valor obtido através do gestor e bloco de comportamento ***eMaginBB***, que soma um total de ~**54ms**, e o valor teórico devido à contabilização no primeiro do **tempo de *polling***, que estava configurado em ***eMaginBB*** com um valor de ***30ms***. Há igualmente um acréscimo de **12ms** do valor obtido pelo facto de o gestor estar inserido dentro do ambiente *Virtools* aumentando assim o tempo total de obtenção dos dados.

7 Testes de Utilizador

7.1 Objectivos

Com os testes de utilizador pretende-se verificar o grau de imersão e interactividade dos jogos desenvolvidos.

7.2 Descrição do sistema a testar

O utilizador irá testar os dois jogos *PianoVR* e *PongVR*. Para isso utiliza um computador com a plataforma *Virtools* devidamente instalada com os seguintes *plugins* desenvolvidos pelo aluno:

- ***GloveManager***, comunicação com luvas
- ***Precision Timer***, utilização de relógio de precisão
- ***TrackerPlugin***, comunicação com *tracker* magnético
- ***VRManager***, comunicação com giroscópio de óculos

Estarão conectados ao computador os seguintes dispositivos:

- ***Emagin Z800 3D Visor***
- ***5DT Data Goves 5 Ultra***
- ***5DT Data Goves 14 Ultra***
- ***Polhemus Fastrak***

7.3 Ambiente de teste

Todos os testes são feitos no gabinete 532 localizado no INESC, Instituto de Engenharia de Sistemas e Computadores, e dado que é neste que se encontram localizados todos os dispositivos imersivos que vão ser utilizados como um computador com a plataforma *Virtools* actualizado com os *plugins* necessários.

7.4 Características dos participantes

Os utilizadores são pessoas do ramo de Informática e Electrotécnica com conhecimentos de programação. No entanto apenas dois destes têm experiência de criação de videojogos.

Nenhum dos participantes tem experiência na utilização dos dispositivos imersivos utilizados neste trabalho.

7.5 Metodologia

Antes dos testes serem efectuados é feita uma pequena explicação do funcionamento de todos os dispositivos, demonstrando o seu uso através do respectivo *software* oficial.

Os utilizadores deverão testar e familiarizar-se com os dispositivos fora dos jogos para que possam ter uma base de comparação ao testar os jogos.

De seguida o utilizador irá começar os testes de utilizador. Para cada um dos jogos é feita uma pequena apresentação de como os dispositivos imersivos foram integrados nos jogos e é feita uma demonstração da sua utilização.

No próximo passo os utilizadores irão calçar os dispositivos necessários para cada jogo e testá-lo. Durante todo o processo de teste é solicitado ao utilizador que fale enquanto esta a testar o jogo sobre:

- O que está a pensar
- O que está a tentar fazer
- Quais as dificuldades que se está a deparar

No final do processo de teste é feito um questionário de modo a quantificar a opinião do utilizador em relação aos jogos

7.6 Tarefas

As tarefas que são efectuadas pelo utilizador são as seguintes:

- Efectuar calibração dos óculos ***3D Visor*** com ***NVIDIA Medical Test***.
- Visualizar ***Demo Test NVIDIA***
- Utilizar/Ambientar a ***Polhemus Fastrak*** com ***software FGUI***
- Utilizar/Ambientar a ***5DT Data Gloves*** com software ***GloveManager***
- Calçar luvas ***5DT 5 Ultra Left Handed***, óculos ***3D Visor*** e um receptor ***Polhemus Fastrak***
- Jogar ***PongVR***
- Responder a questionário
- Calçar ***luvas 5DT 5 Ultra Left Handed*** e ***5DT 14 Ultra Right Handed***, óculos ***3D Visor*** e dois receptores ***Polhemus Fastrak***
- Jogar ***PianoVR***
- Responder a questionário

7.7 Testes e Medidas

Como foi dito anteriormente é pedido aos utilizadores que, no final das tarefas dêem uma classificação de 1-10 a cada um dos tópicos colocados no questionário abaixo para ambos os jogos.

Os valores apresentados no questionário indicam a média aritmética de todos os testes de utilizador efectuados

Tabela 7-1 Questionário *PongVR*

***PongVR* Total**	**Resposta [1-10]**
Sensibilidade de *tracker*	8,5
Tempo de resposta do *tracker*	8,8
Integração das funcionalidades do *tracker* no contexto do jogo	9,0
Sensibilidade de giroscópio óculos	6,8
Tempo de resposta de giroscópio óculos	6,8
Integração das funcionalidades do de giroscópio óculos no contexto do jogo	8,0
Noção de profundidade	7,0
Imersão no jogo	7,5
Interactividade do jogo	8,2
Facilidade do jogo	1,7
Avaliação global do jogo	7,8

Tabela 7-2 Questionário *PianoVR*

***PianoVR* Total**	**Resposta [1-10]**
Sensibilidade de *tracker*	9,2
Tempo de resposta do *tracker*	8, 7
Integração das funcionalidades do *tracker* no contexto do jogo	9,2
Sensibilidade de luvas	8,2
Tempo de resposta de luvas	8,3
Integração das funcionalidades das luvas no contexto do jogo	8,5
Sensibilidade de giroscópio óculos	6,8
Tempo de resposta de giroscópio óculos	7,0
Integração das funcionalidades do de giroscópio óculos no contexto do jogo	7,8
Noção de profundidade	6,8
Imersão no jogo	7, 7
Interactividade do jogo	8,5
Facilidade do jogo	2,5
Avaliação global do jogo	8,2

7.8 Conclusões para Testes de Utilizador

7.8.1 *PongVR*

Observando os resultados obtidos, pode-se concluir que os utilizadores gostaram de jogar e que as funcionalidades disponibilizadas pelo dispositivos foram integradas com sucesso neste jogo. No entanto, queixaram-se do grau de dificuldade do jogo e de, apesar da boa sensibilidade e tempo de resposta do dispositivo de *tracking* conjuntamente com as ajudas visuais e os óculos estereoscópicos, era difícil acertar na bola.

Em relação à utilização do giroscópio dos óculos, todos os utilizadores afirmaram que estes tinham tempos de resposta razoáveis. De acordo com esta opinião, o Gestor dos óculos 3D Visor e bloco de comportamento ***eMaginBB*** foram modificados de modo a se poder alterar o **tempo de *polling*,** dado que esta funcionalidade não estava disponível na primeira versão do bloco de comportamento. O tempo de *polling* define a periodicidade da função de ***CALLBACK*** responsável por actualizar os dados de orientação provenientes do giroscópio interno dos óculos. Com esta modificação pôde-se alterar na aplicação, através do bloco de comportamento ***eMaginBB***, o ***tempo de polling*** de ***100ms***, valor por defeito, para um valor mínimo de ***30ms*** melhorando assim o tempo de resposta dos óculos de ~**123ms** para um valor de ~**54ms**.

A percepção de estereoscopia, ou seja, um bom efeito de profundidade foi comentada como boa. No entanto para cada utilizador esta teve impactos ligeiramente diferentes pelo que, isto deve-se ao factor de apenas se ter calibrado a distância inter pupilar para cada utilizador. Não foi efectuada nenhuma calibração em relação à ao campo de visão da câmara e distância entre esta e o plano de convergência. Tendo esse aspecto em conta foi posteriormente desenvolvida a possibilidade de calibrar ambos estes parâmetros em tempo real durante a execução do jogo para que cada utilizador consiga obter uma maior experiência estereoscópica possível.

7.8.2 *PianoVR*

Para este jogo, apesar de todos os utilizadores terem gostado da sua jogabilidade, houve um maior impacto e agrado nas pessoas com experiência musical. A facilidade de jogo foi, como se pode observar pelos resultados obtidos, superior ao jogo *PongVR*, devendo-se ao facto de neste não ser exigido ao utilizador um tempo de resposta tão rápido aos estímulos provenientes do jogo.

Os resultados obtidos para a utilização de *tracking* e óculos foram praticamente idênticos aos de *PongVR* dado que a sua arquitectura interna é semelhante. Tal como em *PongVR* foi posteriormente desenvolvida a possibilidade de calibrar o campo de visão da câmara e distância entre esta e o plano de convergência em tempo real.

A utilização das luvas foi avaliada como muito boa, as suas funcionalidades foram bem empregues no contexto do jogo, com uma resposta rápida entre os movimentos aplicados na luva e os movimentos.

8 Conclusões

O objectivo principal deste trabalho visa o desenvolvimento de jogos utilizando ferramentas profissionais e dispositivos imersivos estado de arte.

Todo o trabalho realizado na criação de estruturas para a utilização dos dispositivos em *Virtools* é feito de tal forma que possa ser reutilizado em diversos contextos e aplicações. No caso desta tese foi escolhido um contexto lúdico na área de programação de jogos, mas pode ser apontado para aplicações industriais, *p.e.* uma linha de montagem de carros virtual em que o utilizador tem à sua disposição um armazém de peças e consegue agarra-las e as deslocar para uma determinada posição e com uma determinada orientação de modo a construir virtualmente um protótipo de um carro.

É necessário focar toda a ausência de experiência tanto na utilização da ferramenta de desenvolvimento *Virtools* como na utilização dos diversos dispositivos imersivos. Pelo que, no início do desenvolvimento não haver ajuda e suporte no estabelecimento de ensino em que foi desenvolvida dado ser a primeira vez que estava a ser dada uma utilização, tanto para os dispositivos como para o *software Virtools*. Assim há um esforço inicial de auto-aprendizagem tanto da plataforma *Virtools*, utilização do seu *SDK* para o desenvolvimento eficiente de *plugins*, como da correcta utilização de cada um dos dispositivos e respectivas *API's*. Foi também importante aprendizagem de ferramentas de modelação 3D, *Blender*, *3D Studio MAX*, e manipulação de imagem, *ADOBE Photoshop*, de modo a criar o conteúdo 3D e 2D dos jogos.

É conseguido com sucesso a criação de dois jogos, que permitem uma experiência imersiva e interactiva ao utilizador sem fazer uso dos dispositivos de entrada e saída convencionais de um computador.

O facto de ter escolhido a vertente de videojogo motivou a aprendizagem de noções de *game design*, que muitas vezes é ignorado na concepção de um jogo. Foi criada uma arquitectura modular para cada jogo, em que cada um tinha gestores responsáveis por lógica de jogo, comportamento físico, etc. A criação de uma interface ao utilizador/jogador que seja apelativa, contenha toda a informação necessária e ao mesmo tempo não o distraia do objectivo do jogo foi também uma preocupação constante. Toda a criação de elementos tais como *HUD*, menus de jogo, opções disponibilizadas, cenário envolvente, foram estudadas de modo a melhor se adequarem ao tipo de jogo. Por exemplo em *PianoVR* os menus são parte constituinte de uma pauta musical enquanto que em *PongVR* são raquetes de ténis de mesa.

Apesar de *PianoVR* ser um jogo de maior dimensão e mais apelativo, é em *PongVR* que foi encontrado um maior desafio na programação da componente de interactividade, isto porque o tempo de resposta do jogador aos estímulos do jogo é menor. É necessário garantir uma elevada taxa de sucesso nas acções que o jogador toma no curto espaço de tempo que tem de modo a criar motivação para continuar no jogo.

No entanto, realça-se que a utilização dos dispositivos é empobrecida por alguns aspectos. Para o manuseamento do tr*acker* magnético é, é necessário definir uma zona muito específica do espaço. O

facto da ligação entre os receptores e o filtro ser cablada, e também no caso das luvas e óculos *VR* ter de haver um ligação entre estes e a porta *USB*, retira mobilidade à pessoa, esta que tem de necessariamente interagir numa área delimitada pelos diversos cabos. Nota-se que, apesar dos tempos de latência serem idênticos em ambas as luvas, os tensores do modelo *5DT 5 Ultra Left Handed* são menos sensíveis que no modelo *5DT 14 Ultra Right Handed*. Juntando o facto de haver uma discrepância entre o número de sensores em cada uma das luvas, nota-se que movimentação dos dedos da mão virtual esquerda é de qualidade inferior à mão direita.

9 Referências

[1] http://www.theswapmeet-forum.com/

[2] Crespo, Rui Gustavo,"*Specification and Description Language*" (http://comp.ist.utl.pt/ec-st/PDFs/SDL.pdf)

[3] http://pt.wikipedia.org/wiki/Wii_Remote

[4] http://wii.nintendo.com/controller.jsp

[5] "*3SPACE™ FASTRAK® USER'S MANUAL*", 2005 (http://polhemus.com/polhemus_editor/assets/FASTRAKRevE.pdf)

[6] http://www.hitl.washington.edu/projects/knowledge_base/virtual-worlds/tracker-faq.html

[7] http://www.inrialpes.fr/sed/6doftracker/

[8] http://www.worldviz.com/vizhelp/WorldViz_Optical_Tracker_(PPT).htm

[9] http://www.vrealities.com/glove.html

[10] http://www.metamotion.com/hardware/motion-capture-hardware-gloves-Cybergloves.htm

[11] http://www.3dconnexion.com/products/3a1d.php

[12] http://www.3dvisor.com

[13] http://www.cs.nps.navy.mil/people/faculty/capps/4473/projects/mag-track/full.html

[14] Raab H. Frederick, Blood B. Ernest, Steiner O. Terry, Jones R. Herbert, "*Magnetic Position and Orientation Tracking System*", *IEEE Trans. Aerosp. Electron. Syst.*, vol. 5, pp. 709–717, 1979.

[15] "*5DT Data Glove Ultra Series User's Manual*", 2004 (http://5dt.com/downloads/dataglove/ultra/5DT%20Data%20Glove%20Ultra%20-%20Manual.pdf)

[16] "*Stereoscopic Imaging*", ECCS 208, 2006 (http://vis.eng.uci.edu/mediawiki/images/a/ab/Stereo_Introduction_Part1_Part2.pdf)

[17] http://en.wikipedia.org/wiki/Stereoscopy

[18] http://en.wikipedia.org/wiki/Anaglyph_image

[19] Guo Jennifer Xichun, Bruce Land, "*Phong Shading and Gouraud Shading*", Cornell University, Center for Advanced Computing (http://www.nbb.cornell.edu/neurobio/land/OldStudentProjects/cs490-95to96/guo/report.html)

[20] http://ati.amd.com/developer/rendermonkey/index.html

[21] Kessenich John, Baldwin Dave, Rost Randi, "*The OpenGL Shading Language*", 3D Labs, 2006 (http://www.opengl.org/registry/doc/GLSLangSpec.Full.1.20.8.pdf)

[22] Bobick Nick, "*Rotating Objects Using Quaternions*", 1998 (http://www.gamasutra.com/features/19980703/quaternions_01.htm)

[23] Bobic Nick, "*Portal Engines and Object-Object Collisions*", Gamasutra, CMP Media 2003 (http://www.gamasutra.com/features/20000330/bobic_02.htm)

[24] http://www.havok.com/

[25] http://www.ode.org/

[26] http://www.ageia.com/

[27] http://www.newtondynamics.com/

[28] http://www.continuousphysics.com/Bullet/

[29] Kwoon Yen Hun, "*The Theory of Stencil Shadow Volumes*", Gamedev.net, 2002 (http://www.gamedev.net/reference/articles/article1873.asp)

[30] http://en.wikipedia.org/wiki/John_Carmack

[31] Van Der Burg, John, "*Building an Advanced Particle System*", Gamasutra, 2000 (http://www.gamasutra.com/features/20000623/vanderburg_pfv.htm)

[32] Vale, Marco, "*Básicos da Animação*", Knowledge Box, Gamedev-pt 2006 (http://gamedev-pt.net/forum.vertopico.php?tid=826)

[33] http://msdn2.microsoft.com/en-us/library/ms644904.aspx

10 Bibliografia

"*eMagin User Guide*", eMagin Corporation,2005

"*Virtools™ 4.0 Online Reference*", Virtools 2006

Almeida Vieira Miguel, "*Detecção de Colisões*", 2005

Callele David, "*Virtools™ Behaviour Libraries – Physics Library*", Dassault Systemes, 2006

Fernandes Lourenço, Fonseca Manuel João da, "*Métodos de Avaliação e Estatística*", IPM 2005

Foley James, Van Dam Andries, Feiner Steven, Hughes John, "*Computer Graphics Principles and Practices Second Edition*", Addison-Wesley Publishing Company, Inc, 1992

Hagland Torgeir, "*A Fast and Simple Skinning Technique*", "*Game Programming Gems I*", 2000, pp 471-475, Charles River Media, Inc, 2000

Maillot, Patrick Giles, "*Using Quaternions for Coding 3D Transformations*", Graphics Gems I, 1998, pp 498-515, Morgan Kaufmann, 1998

Mc Carthy Christopher, Callele David, Krupa Frédérique, "*Virtools User Guide*", Dassault Systemes, 2006

Möller Akenine Tomas, Haines Eric, "*Real-Time Rendering, Second Edition*", pp239-243, A K Peters, Ltd 2002

Nahon David, Philippon Nicolas, Kaci Leïla Aït, Kuntz Sébastien, "*Virtools™ Behaviour Libraries - VR Library/VR Publisher 2.0 User Guide*", Dassault Systemes, 2006

Rost J. Randi, "*OpenGL Shading Language, Second Edition*", Addison Wesley, 2006

Shreiner Dave, Woo Mason, Neider Jackie, Davis Tom, "*OpenGL Programming Guide, Fifth Edition*", Addison Wesley, 2005

Svarovsky, Jan, "*Quaternions for Game Programming*", Game Programming Gems I, 2000, pp195-199, Charles River Media, Inc, 2000

Taylor II Russell M., Hudson Thomas C., Seeger Adam, Weber Hans, Helser Jeffrey Juliano, Aron T, "*VRPN: A Device-Independent, Network-Transparent VR Peripheral System* ", University of North Carolina

11 Anexos

11.1 Anexo A – Manual de Instruções de jogo *PongVR*

Requisitos Mínimos Hardware:

Pentium III ou equivalente

512 MB RAM

Monitor com resolução de pelo menos 800x600 com 16bit por pixel

Placa gráfica NIVIDIA com suporte a OpenGL ou Direct3D com 64MB de RAM e estereoscopia activa

Rato ou dispositivo semelhante

Drive de CD-ROM

Polhemus Fastrak com um emissor e um receptor

Óculos eMagin Z800 3D Visor

Luva 5DT 5 Ultra Left Handed

Requisitos Mínimos Software:

Microsoft Windows (2000,XP)

Microsoft DirectX 9.0c para placas gráficas que sejam compatíveis com DirectX

OpenGL 2.0 para placas gráficas que sejam compatíveis com OpenGL

Microsoft Internet Explorer 6.0

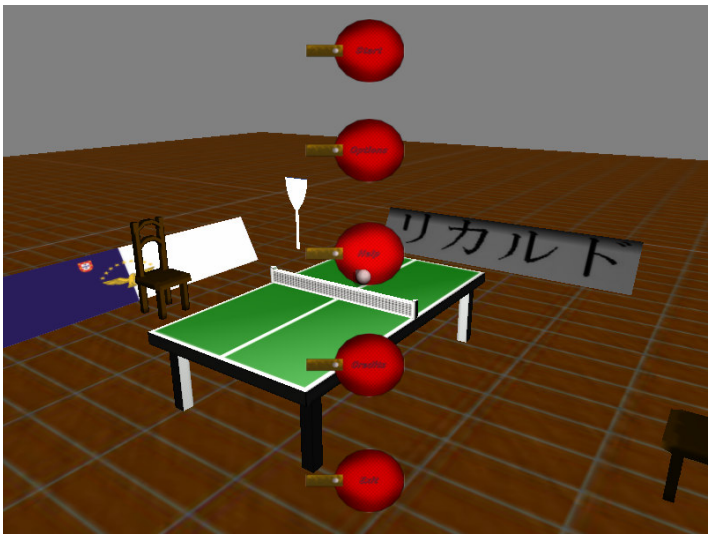
Drivers de suporte a Estereoscopia Activa

Para jogar o jogo Pong VR é necessário antes de o executar ter o seguinte Hardware ligado:

- Tracker Magnético Polhemus Fastrak com um Emissor e Receptor ligados ao Filtro
- Luva 5DT 5 Ultra Left Handed
- Óculos eMagin Z800 3D Visor

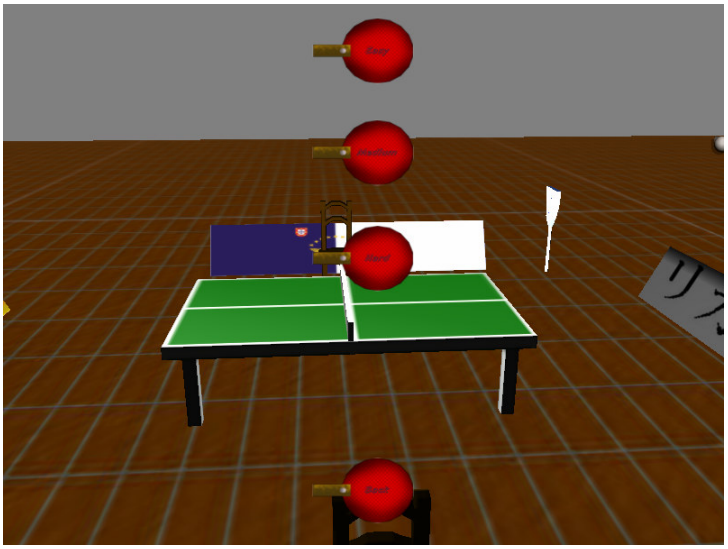
O objectivo do jogador é de ganhar três Sets ao adversário, em que cada Set é ganho caso se consiga obter um total de onze pontos. O jogador ganha um ponto fazendo com que a bola toque consecutivamente no seu campo e no campo do adversário sem que este consiga ripostar.

Ao entrar no jogo, o jogador é confrontado com o seguinte ecrã inicial.



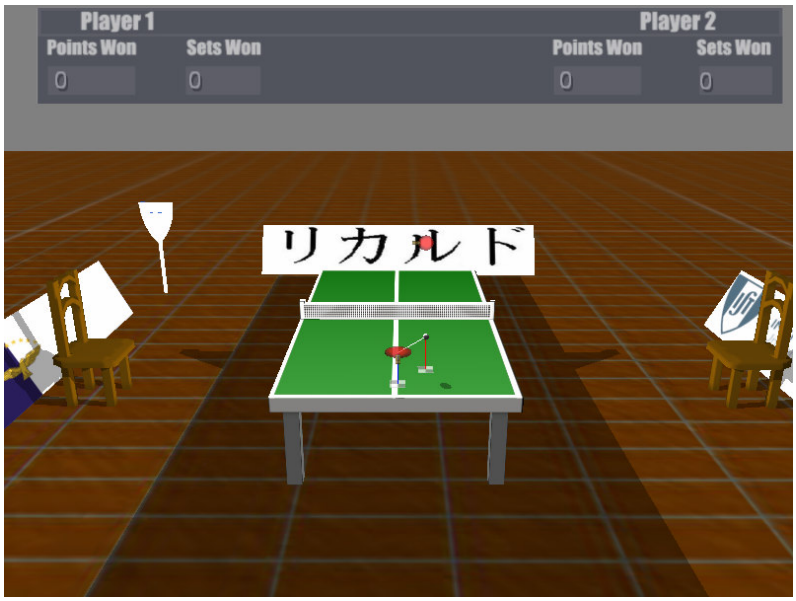
Anexo - Figura 1 *PongVR Main Menu*

- *Start Game*: Primeiro passo para iniciar jogo, passa para menu de jogo para a escolha de dificuldade da IA do adversário.



Anexo - Figura 2 *PongVR Start Menu*

  - *Easy*: Inicia jogo com IA de adversário fácil
  - *Medium*: Inicia jogo com IA de adversário médio
  - *Hard*: Inicia jogo com IA de adversário difícil

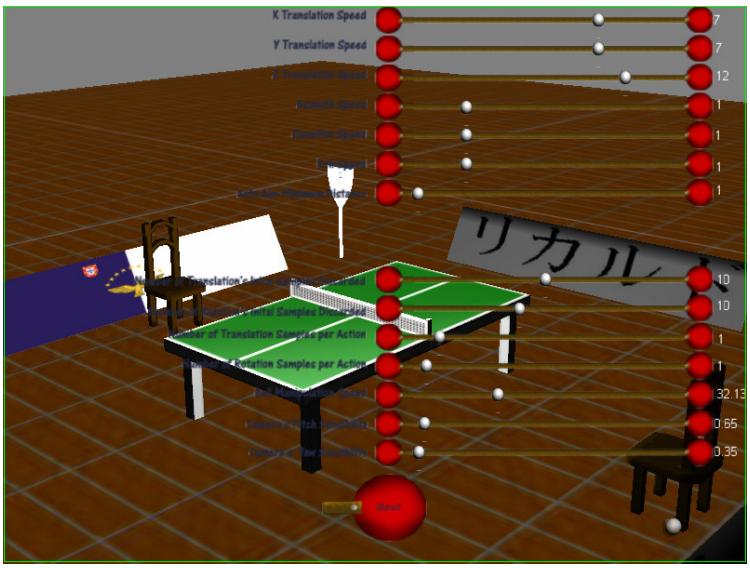


Anexo - Figura 3 *PongVR In Game*

Ao iniciar o jogo é necessário ter cuidado com a posição e orientação tanto do emissor como do receptor, no menu de ajuda é dada informação respeitante a esse assunto.

O *Heads-Up Display* de *PongVR* é constituído por quatro zonas, *Player 1 Points Won*, *Player 1 Sets Won*, *Player 2 Points Won* e *Player 2 Sets Won* que indicam respectivamente o pontos ganhos pelo jogador 1, sets ganhos pelo jogador 1, pontos e sets ganhos pelo jogador 2 (PC).

- *Options*: Abre Menu de Opções



Anexo – *Figura 4 PongVR Options Menu*

- *X Translation Speed* – Sensibilidade no eixo x do receptor de *tracking*
- *Y Translation Speed* – Sensibilidade no eixo y do receptor de *tracking*
- *Z Translation Speed* – Sensibilidade no eixo y do receptor de *tracking*

- *Azimuth Speed* - Sensibilidade no ângulo *azimuth* do receptor de *tracking*
- *Elevation Speed* - Sensibilidade no ângulo *Elevation* do receptor de *tracking*
- *Roll Speed*.-. Sensibilidade no ângulo *Roll* do receptor de *tracking*
- *Auto Aim Min Dist* – Distância para o qual é feito o *auto-aim* da raquete do jogador
- *Ball Magnet Volume* – Factor de escala para Volume de atracção da bola
- *Number of Translation Initial Samples Discarded* – Número de amostras de posição do receptor desprezadas no inicio do jogo
- *Number of Rotation Initial Samples Discarded* – Número de amostras de rotação do receptor desprezadas no inicio do jogo
- *Number of Translation Samples Per Action* - Número de amostras de posição do receptor utilizadas no filtro passa-baixo
- *Number of Rotation Samples Per Action* - Número de amostras de rotação do receptor utilizadas no filtro passa-baixo
- *Ball Manipulation Speed* – Velocidade de translação da bola para posicionamento inicial desta
- *Camera's Pitch Sensitivity* – Sensibilidade do ângulo de *Pitch* aplicado à câmara
- *Camera's Yaw Sensitivity* – Sensibilidade do ângulo de *Yaw* aplicado à câmara

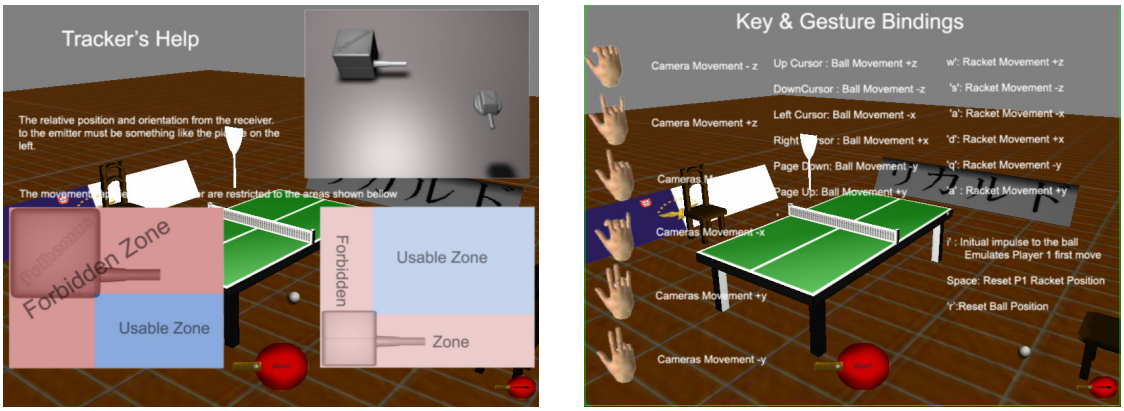
- *Help* – Menu de Ajuda Rápida

No primeiro menu de ajuda é aconselhado a orientações e posições tanto do emissor como do receptor através da figura à esquerda.

Caso não se respeite esta convenção, o movimento e orientação da raquete no cenário não será o esperado.

Após o correcto posicionamento do emissor é necessário ter em conta que a movimentação do receptor deverá ser feita apenas no volume especificado novamente na figura esquerda.

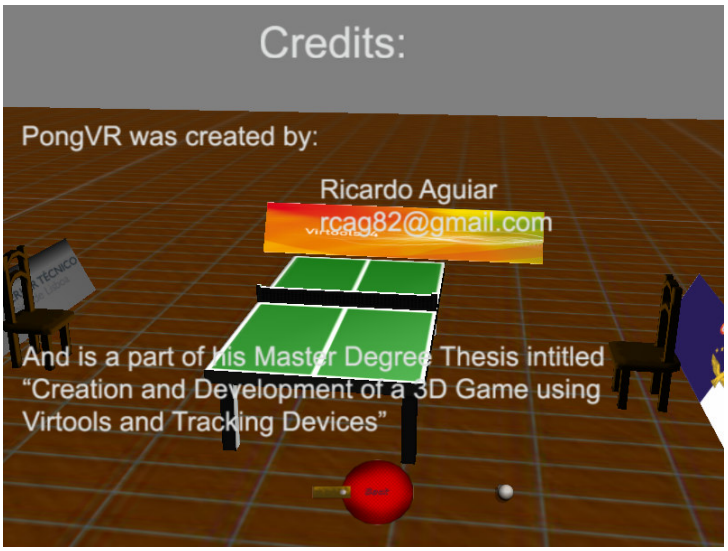
Em relação ao segundo menu de ajuda, este especifica quais os gestos mapeados para o controlo da câmara de jogo e é indicado o mapeamento de teclado.



Anexo - Figura 5 *PongVR Help Menus*

- 'w' - Mover câmara em z+
- 's' - Mover câmara em z-
- 'q' - Mover câmara em y+
- 'z' - Mover câmara em y-
- 'd' - Mover câmara em x+
- 'a' - Mover câmara em x-
- '0' - Aumentar campo de visão
- '9' - Diminuir campo de visão
- '8' - Aproximar plano de convergência de estereoscopia
- '7' - Afastar plano de convergência de estereoscopia

- *Credits* – Créditos do Jogo *PongVR*


Credits:
PongVR was created by:
Ricardo Aguiar
rcag82@gmail.com
And is a part of his Master Degree Thesis intitled
"Creation and Development of a 3D Game using
Virtools and Tracking Devices"

Anexo – Figura 6 ***PongVR Credits Menu***

- *Exit* – Sai do jogo

11.2 Anexo B – Manual de Instruções de jogo PianoVR

Requisitos Mínimos Hardware:

Pentium III ou equivalente

512 MB RAM

Monitor com resolução de pelo menos 800x600 com 16bit por pixel

Placa gráfica NIVIDIA com suporte OpenGL ou Direct3D com 64MB de RAM e estereoscopia activa

Rato ou dispositivo semelhante

Drive de CD-ROM

Polhemus Fastrak com um emissor e um receptor

Óculos *eMagin Z800 3D Visor*

Luva *5DT 5 Ultra Left Handed*

Luva *5DT 14 Ultra Right Handed*

Requisitos Mínimos Software:

Microsoft Windows (2000,XP)

Microsoft DirectX 9.0c para placas gráficas que sejam compatíveis com DirectX

OpenGL 2.0 para placas gráficas que sejam compatíveis com OpenGL

Microsoft Internet Explorer 6.0

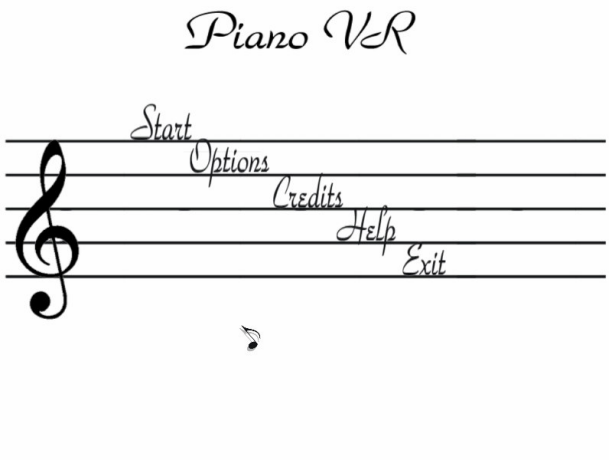
Drivers de suporte a Estereoscopia Activa

Para jogar o jogo Piano VR é necessário antes de o executar ter o seguinte Hardware ligado:

- Tracker Magnético Polhemus Fastrak com um Emissor e dois Receptor ligados ao Filtro
- Luva *5DT 5 Ultra Left Handed* e *5DT 14 Ultra Right Handed*
- Óculos *eMagin Z800 3D Visor*

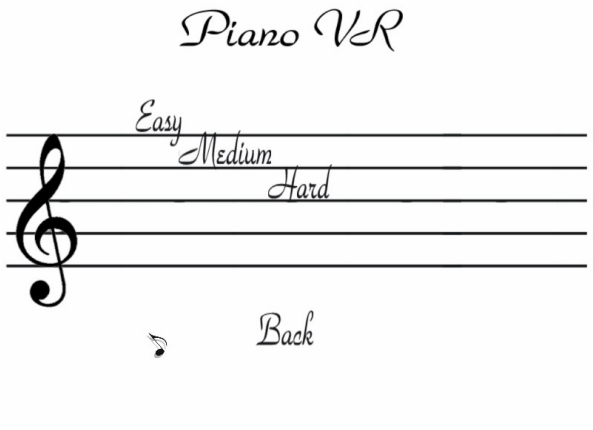
O objectivo do jogo PianoVR é o de completar uma sequência de músicas, cada uma desta com um tempo limite de execução dado pelo grau de dificuldade escolhido.

Ao entrar no jogo, o jogador é confrontado com o seguinte ecrã inicial.



Anexo - Figura 7 PianoVR Main Menu

- Start Game: Passa para menu de jogo para a escolha de dificuldade da dificuldade do jogo



Anexo - Figura 8 *PianoVR Start Menu*

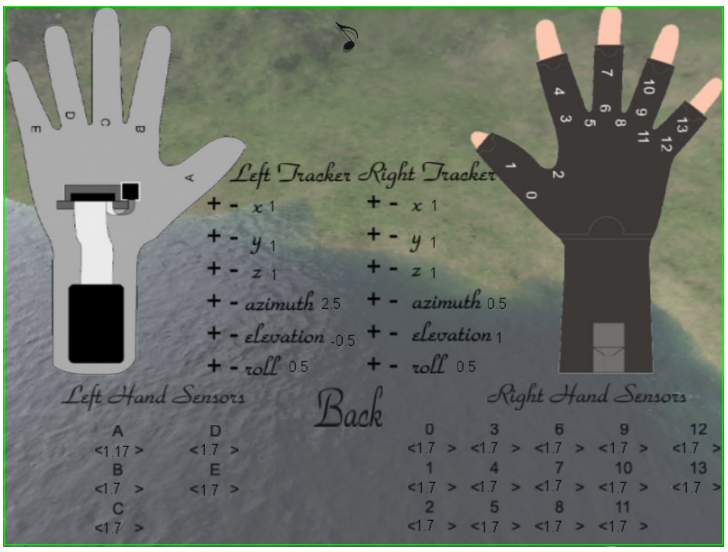
- Easy: Inicia jogo com tempo de execução de 4 minutos por música, fácil
- Medium: Inicia jogo com tempo de execução de 2 minutos por música, médio
- Hard: Inicia jogo com tempo de execução de 1 minuto por música, difícil

Anexo - Figura 9 *PianoVR In Game*

O *Heads-Up Display* de jogo é constituído por quatro zonas, *Next Note* em que é dada informação numa Clave de Sol de qual a próxima nota a ter de ser tocada, *Success Rate* que indica a percentagem de notas tocadas na sequência correcta*, Remaining Time* que indica o tempo restante para tocar a sequência de notas para a música actual e *Elapsed Time* que indica o tempo total de jogo.

É também dada informação visual através da cor azul nas teclas para saber qual a próxima tecla a carregar de modo a efectuar correctamente a sequência para a música actual.

- *Options*: Abre Menu de Opções



Anexo - Figura 10 *PianoVR Options Menu*

- *Left Tracker*:
  - X – Sensibilidade no eixo x do receptor de tracking esquerdo
  - Y – Sensibilidade no eixo y do receptor de tracking esquerdo

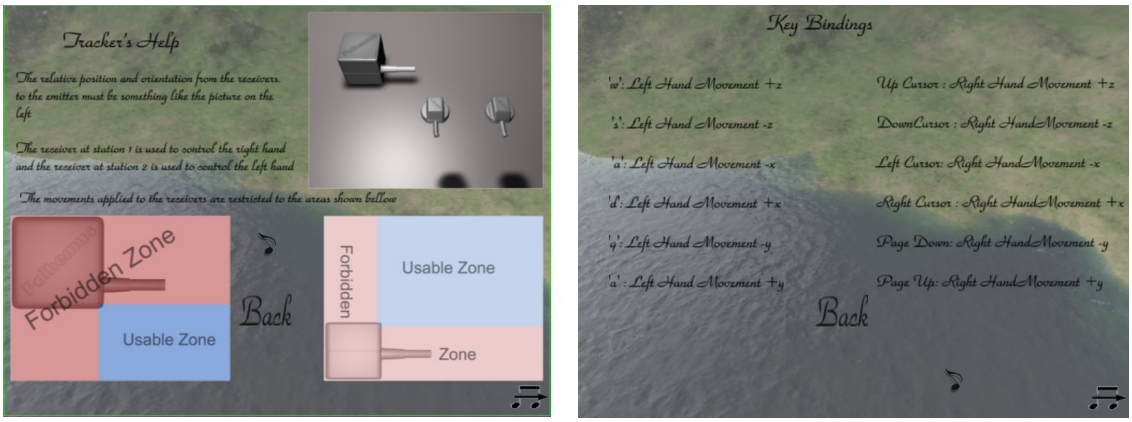
        - Z – Sensibilidade no eixo y do receptor de tracking esquerdo
        - *Azimuth* – Sensibilidade no ângulo azimuth do receptor de tracking esquerdo
        - *Elevation* – Sensibilidade no ângulo Elevation do receptor de tracking esquerdo
        - *Roll*.-. Sensibilidade no ângulo Roll do receptor de tracking esquerdo
    - *Right Tracker*:
        - X – Sensibilidade no eixo x do receptor de tracking direito
        - Y – Sensibilidade no eixo y do receptor de tracking direito
        - Z – Sensibilidade no eixo y do receptor de tracking direito
        - *Azimuth* – Sensibilidade no ângulo azimuth do receptor de tracking direito
        - *Elevation* – Sensibilidade no ângulo Elevation do receptor de tracking direito
        - *Roll*.-. Sensibilidade no ângulo Roll do receptor de tracking direito
    - *Left Hand Sensors* [A – D] – Sensibilidade dos sensores [A-D → ver figura] da luva da mão esquerda
    - *Right Hand Sensors* [0 -13] – Sensibilidade dos sensores [0-13 → ver figura] da luva da mão direita
    -

- *Help* – Menu de Ajuda Rápida

Ao iniciar o jogo é necessário ter cuidado com a posição e orientação tanto do emissor como dos dois receptor. A posição relativa de ambos os sensores é irrelevante dado que a aplicação faz um ajuste automático de modo a que a distância real entre os receptores seja correspondente à distância real.

É aconselhado as seguintes orientações e posições relativas do emissor como os dois receptores através da figura à esquerda. Caso não se respeite esta convenção, o movimento e orientação da raquete no cenário não será o esperado.

Após o correcto posicionamento do emissor é necessário ter em conta que a movimentação do receptor só deverá ser feita no volume especificado novamente especificado pela figura abaixo.

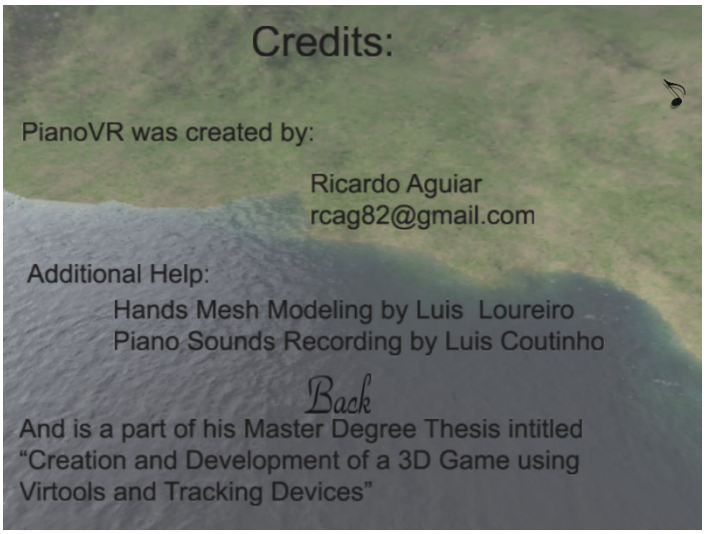

Tracker's Help
The relative position and orientation from the receivers to the emitter must be something like the picture on the left
The receiver at station 1 is used to control the right hand and the receiver at station 2 is used to control the left hand
The movements applied to the receivers are restricted to the areas shown bellow
Forbidden Zone
Usable Zone
Back
Forbidden
Usable Zone
Zone
Key Bindings
'w': Left Hand Movement +z
's': Left Hand Movement -z
'a': Left Hand Movement -x
'd': Left Hand Movement +x
'q': Left Hand Movement -y
'a': Left Hand Movement +y
Up Cursor : Right Hand Movement +z
DownCursor : Right HandMovement -z
Left Cursor: Right HandMovement -x
Right Cursor : Right HandMovement +x
Page Down: Right HandMovement -y
Page Up: Right HandMovement +y
Back

Anexo - Figura 11 ***PianoVR Help Menus***

Para além das teclas indicadas no menu de ajuda estão também mapeadas as seguintes funcionalidades:

- 't' - Mover câmara em z+
- 'g' - Mover câmara em z-
- 'r' - Mover câmara em y+
- 'c' - Mover câmara em y-
- 'h' - Mover câmara em x+
- 'f' - Mover câmara em x-
- '0' - Aumentar campo de visão
- '9' - Diminuir campo de visão
- '8' - Aproximar plano de convergência de estereoscopia
- '7' - Afastar plano de convergência de estereoscopia

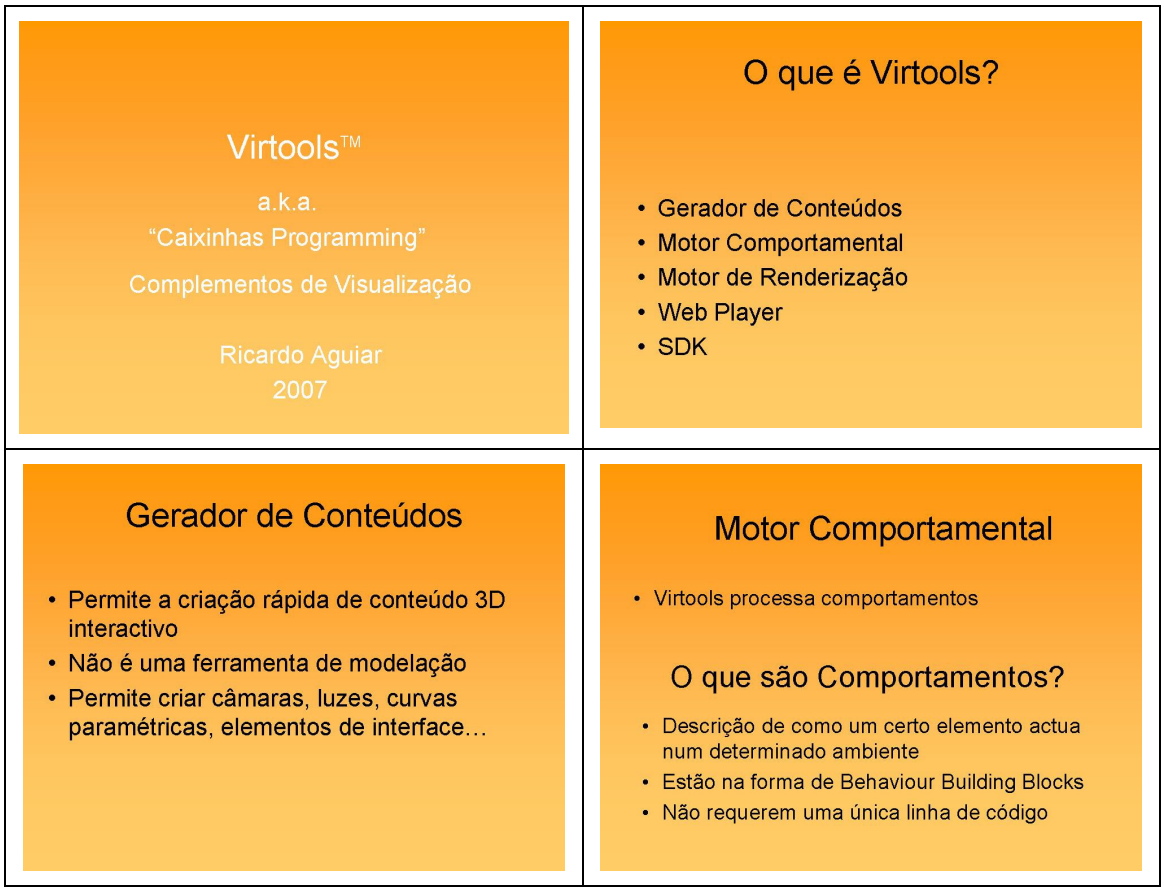
- Credits – Créditos do Jogo *PianoVR*


Credits:
PianoVR was created by:
Ricardo Aguiar
rcag82@gmail.com
Additional Help:
Hands Mesh Modeling by Luis Loureiro
Piano Sounds Recording by Luis Coutinho
Back
And is a part of his Master Degree Thesis intitled
"Creation and Development of a 3D Game using
Virtools and Tracking Devices"

Anexo - Figura 12 *PianoVR Credits Menu*

- *Exit* – Sai do jogo

11.3 Anexo C - Apresentação *Virtools*


Virtools™
a.k.a.
"Caixinhas Programming"
Complementos de Visualização
Ricardo Aguiar
2007
O que é Virtools?
• Gerador de Conteúdos
• Motor Comportamental
• Motor de Renderização
• Web Player
• SDK
Gerador de Conteúdos
• Permite a criação rápida de conteúdo 3D interactivo
• Não é uma ferramenta de modelação
• Permite criar câmaras, luzes, curvas paramétricas, elementos de interface…
Motor Comportamental
• Virtools processa comportamentos
O que são Comportamentos?
• Descrição de como um certo elemento actua num determinado ambiente
• Estão na forma de Behaviour Building Blocks
• Não requerem uma única linha de código

Motor de Renderização

- Desenha conteúdo no ecrã
- Funciona sobre DX9, DX8, DX7, OpenGL 2.0
- Pode ser substituído por um motor construído pelo utilizador através do SDK

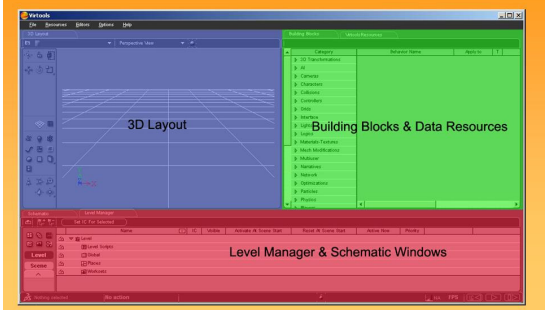
Web Player

- 3D Life Player
- Permite a visualização/interacção de conteúdo através de páginas Web

SDK

- Permite:
  - Criar/Modificar comportamentos
  - Criar novos tipos de parâmetros
  - Criar plugins de media
  - Modificar/Extender/Substituir Motor de Renderização

Layout Virtools



3D Layout

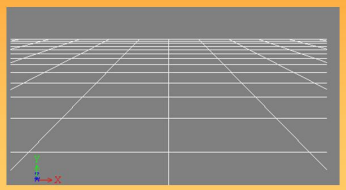
- Constituído por:
  - Barra de Ferramentas no Topo
    - 3D Layout Explorer
    - Selection Group
    - Select Camera
    - General Preferences

3D Layout

- Constituido por:
  - Barra de ferramentas à esquerda
    - Selection Tools
    - Transformations Tools
    - Reference & Screen Guides
    - Creation Tools
    - Camera Navigation Tools

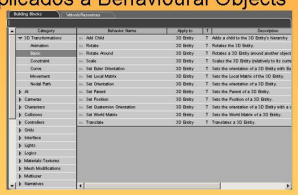
3D Layout

- Constituido por:
  - Janela de Renderização



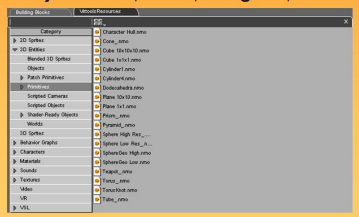
Building Blocks & Data Resources

- Contém comportamentos disponíveis ao utilizador
  - Organizados por tipo
  - Aplicados a Behavioural Objects



Building Blocks & Data Resources

- Recursos Multimédia disponibilizados pela Virtools
  - Objectos 3D, Sons, Imagens, etc…

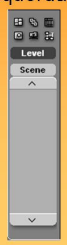


Level Manager & Schematic Windows

- Level Manager consiste:
  - Barra de Ferramentas no topo
    - Mostrar/Ocultar Layers
    - Expandir/Colapsar elementos seleccionados

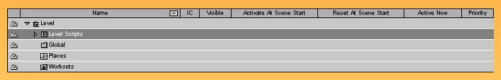
Level Manager & Schematic Windows

- Level Manager consiste:
  - Barra de Ferramentas à esquerda
    - Criação de:
      - Grupos
      - Arrays
      - Scripts
      - Scenes
      - …



Level Manager & Schematic Windows

- Level Manager consiste:
  - Janela Principal
    - Lista todos os elementos do Level/Scene



Schematic Window

- Constituída por :
  - Barra de Ferramentas no topo
    - Diversas funções de ajuda na manipulação de scripts (modo Debug, Mostrar/ Esconder Scripts,etc)

Schematic Window

- Constituída por :
  - Zona de criação e modificação de conteúdo nos scripts
    - "Coração" da nossa composição

Barra de Status

- Localizada no canto inferior do ecrã
- Fornece informação útil sobre a Composição
  - Informação de performance
    - FPS, taxa de ocupação de recursos
  - Coordenadas do Mundo para um dado objecto seleccionado

Schematic Window

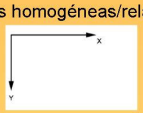
- Constituída por :
  - Zona de criação e modificação de conteúdo nos scripts
    - "Coração" da nossa composição

Barra de Status

- Localizada no canto inferior do ecrã
- Fornece informação útil sobre a Composição
  - Informação de performance
    - FPS, taxa de ocupação de recursos
  - Coordenadas do Mundo para um dado objecto seleccionado

Sistema de Coordenadas na Virtools

- Sistema de Coordenadas 2D
  - Definida por 2 eixos (X Horizontal, Y Vertical) com origem no canto superior esquerdo do ecrã
  - Coordenadas 2D ou Coordenadas de ecrã expressas em:
    - Pixeis, coordenadas absolutas
    - Coordenadas homogéneas/relativas [0.0 – 1.0]



Sistema de Coordenadas no Virtools

- Sistema de Coordenadas 3D
  - Contém Profundidade para além de Altura e Largura
    - Novo Eixo Z – Profundidade
  - Sistema de Coordenadas do Mundo
    - Origem no ponto de intersecção dos 3 eixos
  - Unidades de Medida
    - Sugestão: 1 unidade no Sistema de Coordenadas 3D por defeito corresponde a 1 metro

Sistema de Coordenadas no Virtools

- Sistema de Coordenadas 3D
  - Coordenada 3D é:
    - Posição (x,y,z) num certo Sistema de Coordenadas 3D
  - Sistema de Coordenadas 3D local
    - Cada entidade 3D contém um Sistema de Coordenadas 3D local
      - Canto da Bounding Box correspondente
      - Barycenter

Sistema de Coordenadas no Virtools

- Transformações:
  - Sistema de Coordenadas Local
  - Sistema de Coordenadas do Mundo
    - Usando termo especial --NULL--

O que é uma Composição?

- Mundo Virtual
- Elementos em cena e os seus comportamentos associados
  - Podem ter um presença visual
    - Personagens, objectos cénicos
  - Podem ser elementos abstractos
    - Ligações lógicas, arrays
  - Comportamentos
    - São o que tornam um elemento interactivo
      - Utilizador
      - Outros elementos na Composição

Elementos de uma Composição CMO

- BeObject
  - Render Objects
    - 2D Entity
    - 3D Entity
  - Objectos "invisíveis"
    - Cameras
    - Curves
    - Grids
    - Lights
    - Arrays
    - Groups
    - Level
    - Material
    - Mesh
    - Place
    - Scene
    - Sound
    - Texture

Virtools Process Loop

- Processo repetitivo que ocorre quando um utilizador corre uma composição
- Duração de um Process Loop é chamado de Frame
- Constituído por dois passos
  - Processamento de comportamentos
  - Renderização da cena de Jogo

1 Frame		1 Frame		And so on…
Process Behaviors	Rendering	Process Behaviors	Rendering	

Virtools Process Loop – Passos

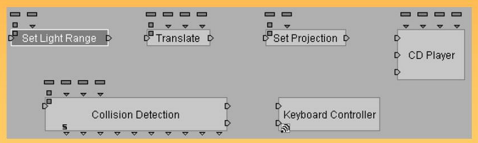
- Processamento de Comportamentos
  - Comportamentos activos/executados por prioridade
  - Um comportamento pode activar outros através de links que contêm um certo atraso
    - 0 – propaga a activação na mesma frame
    - 1 – propaga a activação na próxima frame
    - n – propaga a activação na n^{ésima} frame a seguir à actual

Virtools Process Loop – Passos

- Renderização da cena de Jogo
  - Mostra conteúdo da composição
  - Tarefa mais dispendiosa
  - Dispositivos de Render independentes para modo de execução e modo de criação
  - Dispositivos disponíveis dependentes da placa gráfica

Comportamentos

- Identificados como Behavioural Building Blocks – BB's
- Caixinhas que mudam o Mundo
- Existem em diferentes tipos e tamanhos

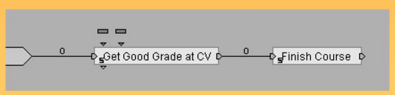


Comportamentos – Behavioural Building Blocks – BB's

- Como interpretar a representação de uma BB?
  - Behaviour Input – bIn:
    - Localizada à esquerda da BB
    - Inicia o processamento da BB
  - Behaviour Output – bOut:
    - Localizada à direita da BB
    - É activada no final do processamento da BB dentro de uma frame
    - Fluxo de activação percorre links que estejam ligados a essa

Comportamentos – Behavioural Building Blocks – BB's

- Como interpretar a representação de uma BB?
  - Behaviour link – bLink:
    - BB ligadas por bLinks
    - Definem sequência de processamento
    - Atraso especificado pelo programador

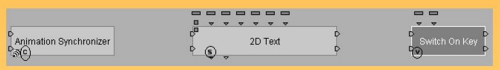


Comportamentos – Behavioural Building Blocks – BB's

- Como interpretar a representação de uma BB?
  - Parâmetros de entrada/saída pIn/pOut
    - Forma de a BB receber e transmitir dados
  - Parâmetro de alvo
    - Tipo especial de parâmetro
    - Indica explicitamente a entidade a ser afectada pela BB

Comportamentos – Behavioural Building Blocks – BB's

- Como interpretar a representação de uma BB?
  - Algumas BB's estão marcadas com as letras C,S e V no seu canto inferior esquerdo
    - C - Custom Dialog Box
    - S - Settings
    - V – Variable configuration

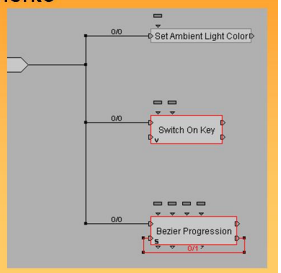


Comportamentos – Behavioural Building Blocks – BB's

- Como interpretar a representação de uma BB?
  - BB podem ter icone de Mensagem
    - Pode enviar ou receber mensagens
    - Diversos usos

Comportamentos – Behavioural Building Blocks – BB's

- Tipos de Processamento
  - Single Action
  - Internally Looped
  - Externally Looped



Comportamentos – Behaviour Graphs

- Composição de BB's encapsuladas
- É tratada como se fosse em si uma BB
- Pode ser salva e reutilizada

Parâmetros

- Constituídos por:
  - Nome – pName
  - Tipo de dados – pType
  - Valor – pValue
- Parameter Input, pIn
  - Argumentos de entrada de um comportamento
  - Representados por um triângulo em cima de uma BB, paramOp ou BG

Parâmetros

- Parameter Output, pOut
  - Valores processados pela BB, paramOp ou BG
- Parameter Link, pLink
  - Propaga o valor de um pOut para um pIn
  - Não contém atraso

Parâmetros

- Parâmetros Locais
  - Guardam Valores até serem necessários
- Parâmetro This
  - Referência ao "Dono" de um determinado script
- Atalhos para parâmetros
  - Simplificação visual
  - Utilização de parâmetros entre scripts

Parâmetros

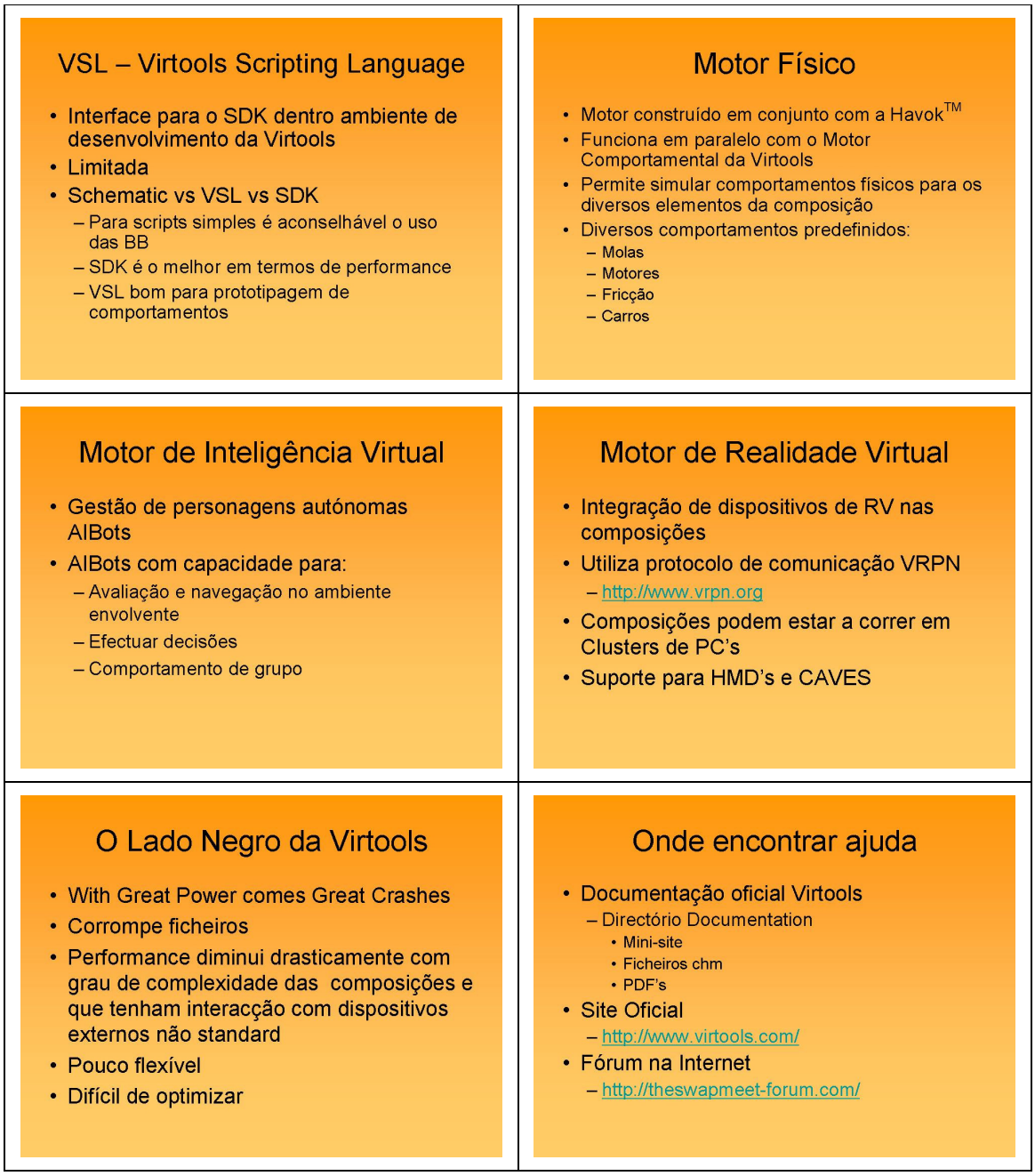
- Operações sobre Parâmetros, ParamOps
  - Operações sobre um ou um par de parâmetros
  - Obter informação do motor de comportamento
  - Efectuar operações matemáticas
  - Conversão de tipo de parâmetro

Parâmetros

- Operações sobre Parâmetros, ParamOps
  - Não são activos da mesma maneira que BB's
  - Só é calculado quando é necessário
    - Pedido por uma BB ou outro ParamOp
  - Aplicados a qualquer tipo de Script
  - Sincronizados com o fluxo de activação de um script
    - BB Op e BB Identify

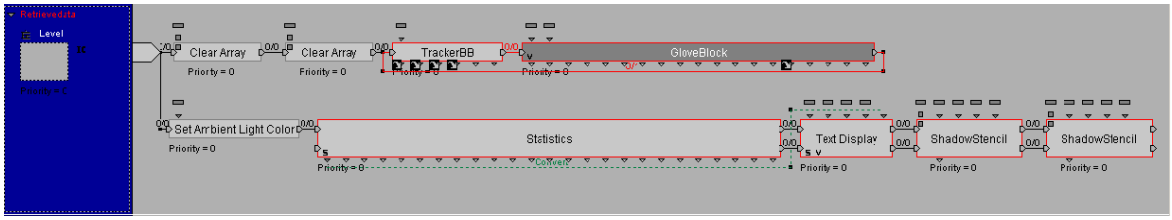
Atributos

- Informação extra de um elemento
- Aplicados apenas as BehObjects
- Contêm um Nome e Categoria
- Podem apenas ser editados em Author Mode
- Gestão de Atributos pelo Attributes Manager

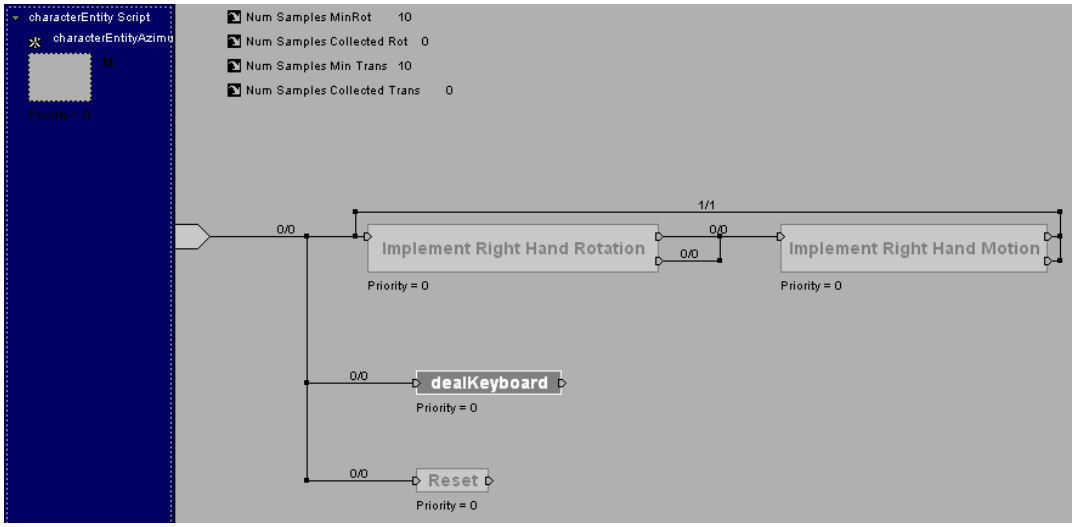

VSL – Virtools Scripting Language
• Interface para o SDK dentro ambiente de desenvolvimento da Virtools
• Limitada
• Schematic vs VSL vs SDK
– Para scripts simples é aconselhável o uso das BB
– SDK é o melhor em termos de performance
– VSL bom para prototipagem de comportamentos
Motor Físico
• Motor construído em conjunto com a Havok™
• Funciona em paralelo com o Motor Comportamental da Virtools
• Permite simular comportamentos físicos para os diversos elementos da composição
• Diversos comportamentos predefinidos:
– Molas
– Motores
– Fricção
– Carros
Motor de Inteligência Virtual
• Gestão de personagens autónomas AIBots
• AIBots com capacidade para:
– Avaliação e navegação no ambiente envolvente
– Efectuar decisões
– Comportamento de grupo
Motor de Realidade Virtual
• Integração de dispositivos de RV nas composições
• Utiliza protocolo de comunicação VRPN
– http://www.vrpn.org
• Composições podem estar a correr em Clusters de PC's
• Suporte para HMD's e CAVES
O Lado Negro da Virtools
• With Great Power comes Great Crashes
• Corrompe ficheiros
• Performance diminui drasticamente com grau de complexidade das composições e que tenham interacção com dispositivos externos não standard
• Pouco flexível
• Difícil de optimizar
Onde encontrar ajuda
• Documentação oficial Virtools
– Directório Documentation
• Mini-site
• Ficheiros chm
• PDF's
• Site Oficial
– http://www.virtools.com/
• Fórum na Internet
– http://theswapmeet-forum.com/

Anexo - Figura 13 Apresentação *Virtools*

11.4 Anexo D – Scripts PongVR

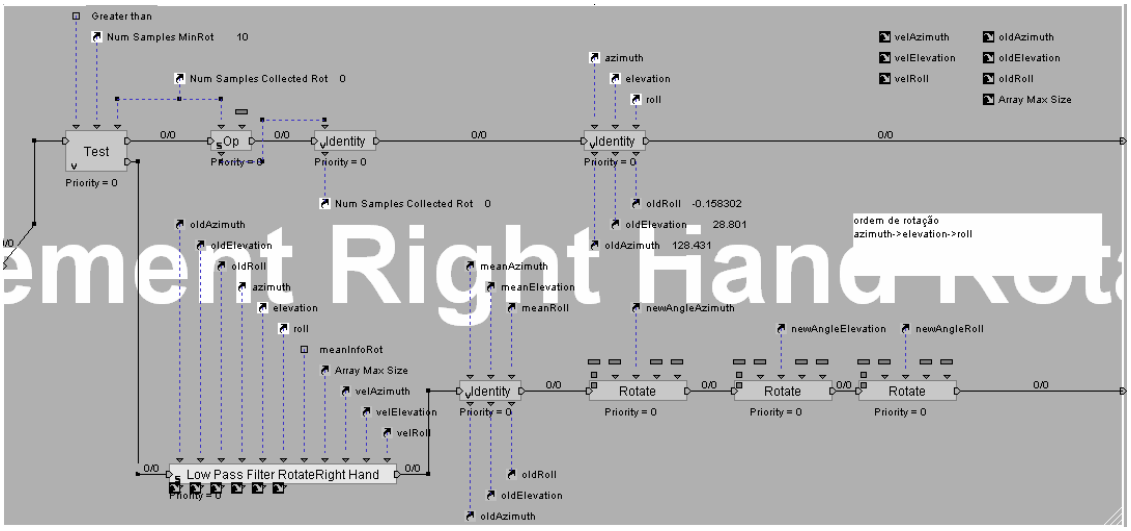


Anexo - Figura 14 Script de comunicação com Gestores

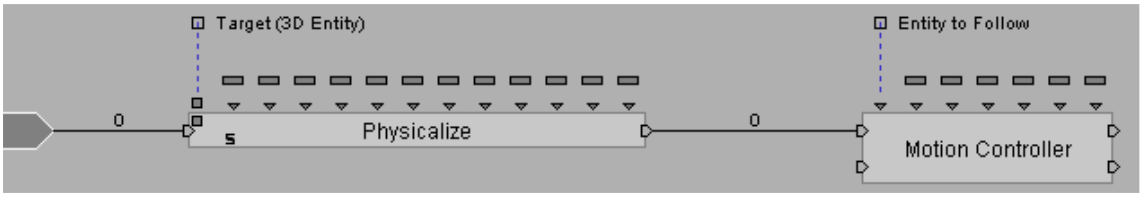


Anexo - Figura 15 Script de tratamento de dados

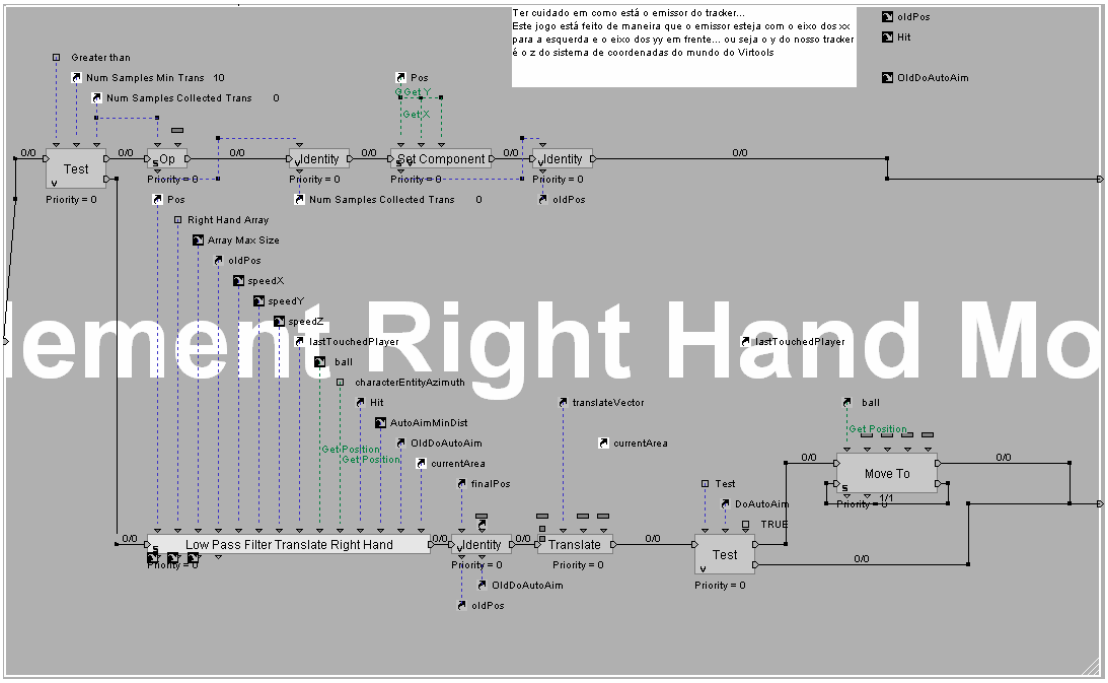
De notar que os comportamentos *LowPass Filter Rotate Right Hand* e *LowPass Filter Translate Right Hand* não são comportamentos nativos da *Virtools*, mas sim *scripts VSL* escritos especificamente para a aplicação



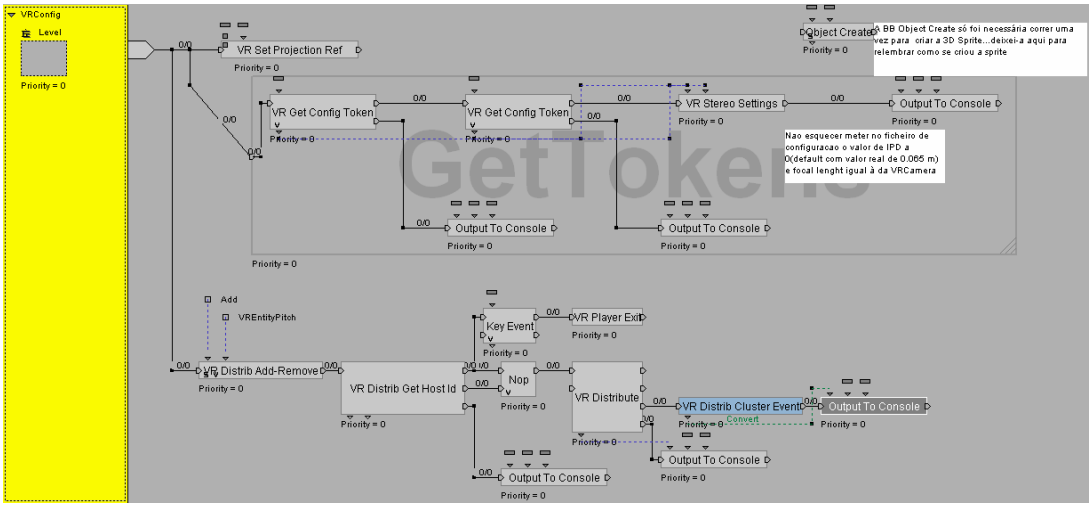
Anexo - Figura 16 Conteúdo grafo comportamental 'Implement Right Hand Rotation'



Anexo - Figura 17 Movimentação de objecto após inclusão no motor de física

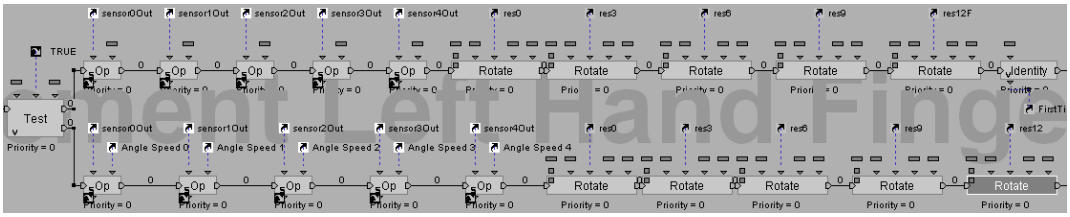


Anexo - Figura 18 Conteúdo grafo comportamental 'Implement Right Hand Motion'


VRConfig
Level
VR Set Projection Ref
Object Create
BB Object Create só foi necessária correr uma vez para criar a 3D Sprite...deixei-a aqui para relembrar como se criou a sprite
VR Get Config Token
VR Get Config Token
VR Stereo Settings
Output To Console
Nao esquecer meter no ficheiro de configuracao o valor de IPD a 0(default com valor real de 0.065 m) e focal lenght igual à da VRCamera
Output To Console
Output To Console
Add
VREntityPitch
Key Event
VR Player Exit
VR Distrib Add-Remove
VR Distrib Get Host Id
Nop
VR Distribute
VR Distrib Cluster Event
Output To Console
Output To Console
Output To Console

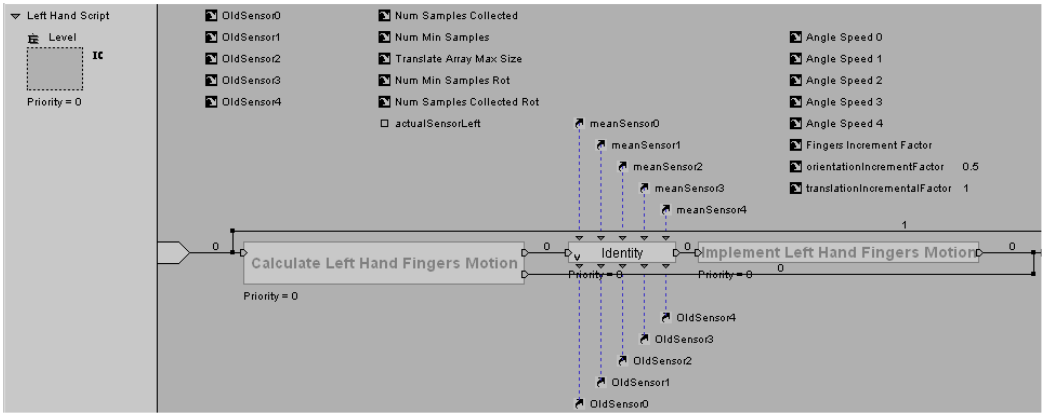
Anexo - Figura 19 Script de configuração de estereoscopia

11.5 Anexo E – Scripts PianoVR

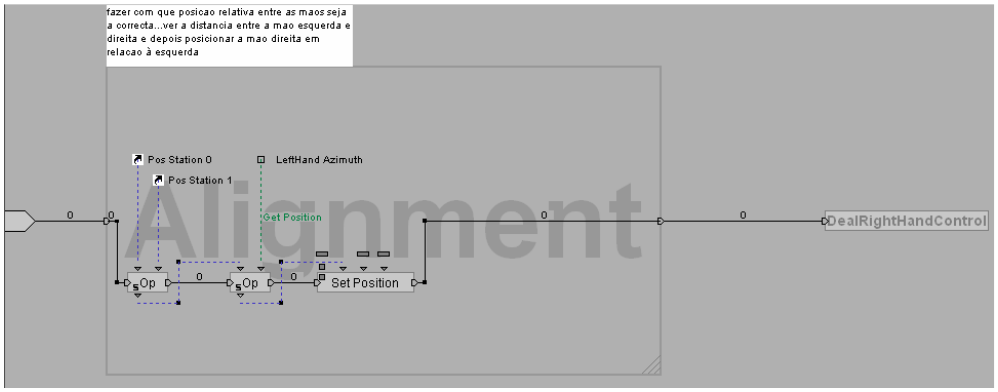


Anexo - Figura 20 Grafo comportamental 'Implement Left Finger Motion'

De notar que *Low Pass Hand Filter* e *New Angles Left Hand* são *scripts VSL* criados especificamente para a aplicação.



Anexo - Figura 21 Script de tratamento de dados de mão esquerda


fazer com que posicao relativa entre as maos seja a correcta...ver a distancia entre a mao esquerda e direita e depois posicionar a mao direita em relacao à esquerda
Pos Station 0
Pos Station 1
LeftHand Azimuth
Get Position
Op
Set Position
DealRightHandControl

Anexo - Figura 22 Alinhamento mão direita